NEURAL NETWORK ENHANCED SELF TUNING ADAPTIVE CONTROL APPLICATION FOR NON-LINEAR CONTROL OF DYNAMIC SYSTEMS

Thesis submitted to the Aston University for the degree of Doctor of Philosophy

by

BAL KRISHNA THAPA

बाल कृष्ण थापा

THE

ASTON UNIVERSITY

June 2001

I dedicate this work to
My Godfather,
Major T.J.W.Allen, M.C.

SUMMARY

The main theme of research of this project concerns the study of neural networks to control uncertain and non-linear control systems. This involves the control of continuous time, discrete time, hybrid and stochastic systems with input, state or output constraints by ensuring good performances.

A great part of this project is devoted to the opening of frontiers between several mathematical and engineering approaches in order to tackle complex but very common non-linear control problems.

The objectives are

1. Design and develop procedures for neural network enhanced self-tuning adaptive non-linear control systems.
2. To design, as a general procedure, neural network generalised minimum variance self-tuning controller for non-linear dynamic plants (Integration of neural network mapping with generalised minimum variance self-tuning controller strategies).
3. To develop a software package to evaluate control system performances using Matlab, Simulink and Neural Network toolbox.

An adaptive control algorithm utilising a recurrent network as a model of a partially unknown non-linear plant with unmeasurable state is proposed. Appropriately, it appears that structured recurrent neural networks can provide conveniently parameterised dynamic models for many non-linear systems for use in adaptive control.

Properties of static neural networks, which enabled successful design of stable adaptive control in the state feedback case, are also identified.

A survey of the existing results is presented which puts them in a systematic framework showing their relation to classical self-tuning adaptive control application of neural control to a SISO / MIMO control.

Simulation results demonstrate that the self-tuning design methods may be practically applicable to a reasonably large class of unknown linear and non-linear dynamic control systems.

Keywords: Feed-forward Neural Network (FFNN), Recurrent Neural Network (RNN), On-line, Off-line, Self-tuning adaptive control, System Identification, Predictive Control, Linear and Non-linear Dynamic Control Systems.

ACKNOWLEDGEMENTS

I am indebted to my supervisor Prof. B.Jones and Dr. Q.M.Zhu (my initial supervisor) for their continuous invaluable support and guidance during the course of this research. Their enthusiasm, friendship and encouragement have greatly contributed to my understanding of the research process while also maintaining my sanity.

I am very grateful to my family especially my elder brother Ghan Bahadur Thapa who gave me encouragement, understanding and support throughout my life.

Last but not least, I wish to extend my deepest gratitude to my godfather, Major T.J.W.Allen, M.C., for his financial support, encouragement and for much more than can be mentioned herein.

I would like to extend my thanks to my friends Eamonn Baldwin, Tim Earthrowl-Gould and Steve Khoo for their help with my English and friendship during my time at Aston University. Also my thanks go to Lingzhong Goa for his help in explaining some programming techniques.

Finally, I am most grateful to those people who gave me valuable feedback and recommendations that made this thesis possible.

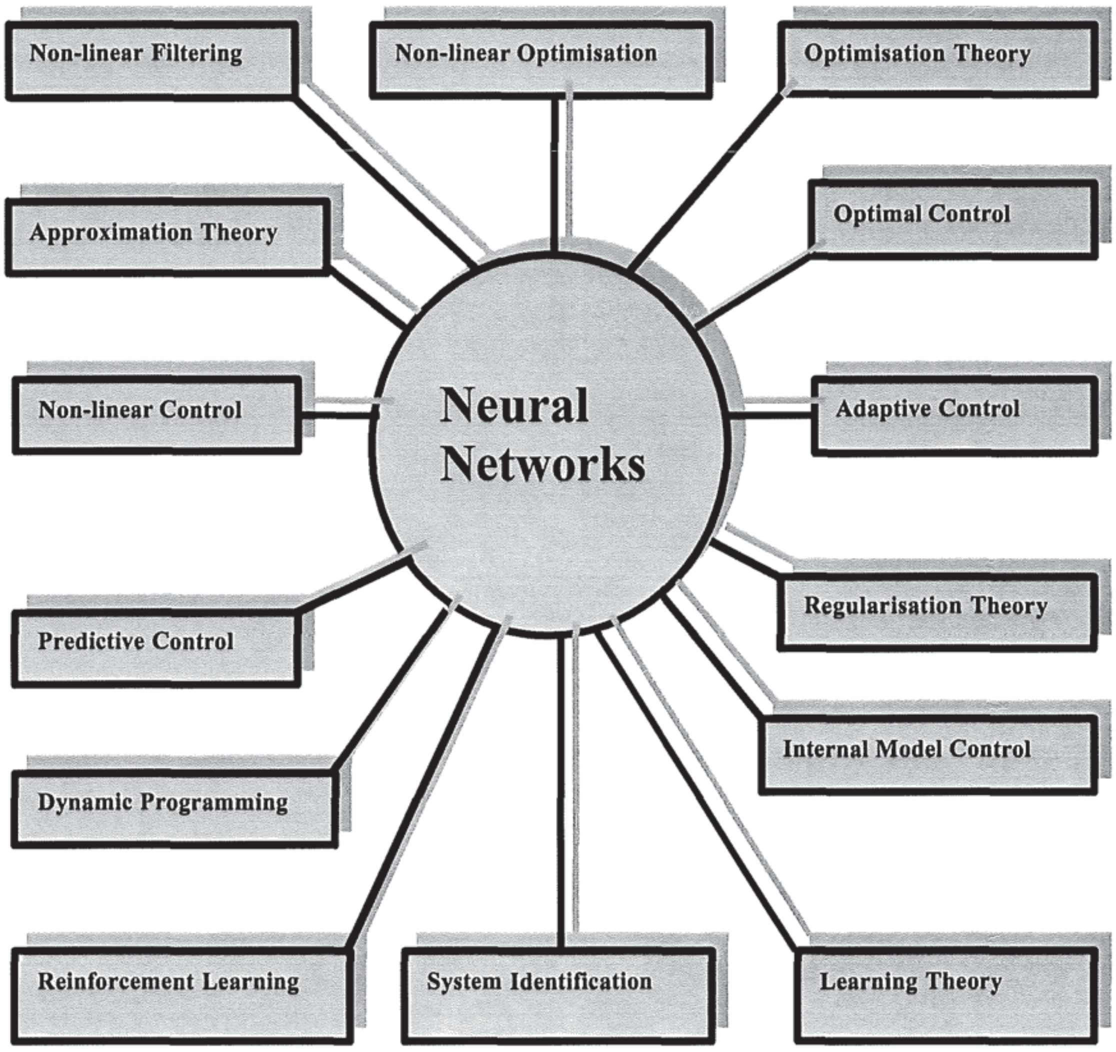


Fig.1 Overview of neural network application areas

LIST OF CONTENTS **Page Numbers**

LIST OF FIGURES Page Numbers

In Chapter 4:

NOTATION

Symbols	Description
α (alpha)	Momentum factor (momentum constant)
η(eta)	Learning rate parameter of neural network
e	Error
E	Error function or cost function
k	Time step
λ(lamda)	Forgetting factor or Scaling constant in reinforcement learning
n	Number of output units unless stated otherwise
n_u	Number of past plant input signals
n_y	Number of past plant output signals
P	Covariance matrix
r *or* d	Reference signal input
S	Sum of input to a unit
t	continuous time
τ_d	Time-delay
z^{-1} *or* q^{-1}	Unit delay operator (backward shift operator)
norm(X)	$\begin{bmatrix} \overline{\sigma}(X_{11})\cdots\overline{\sigma}(X_{1m}) \\ \vdots\vdots \\ \overline{\sigma}(X_{m1})\cdots\overline{\sigma}(X_{mm}) \end{bmatrix} if\, X = \begin{bmatrix} X_{11})\cdots X_{1m}) \\ \vdots\vdots \\ X_{m1})\cdots X_{mm}) \end{bmatrix}$
y_d	Desired output signal
y_m	Actual model output
y_p	Actual plant output
y_r	Reference model output

Special Symbols

d	Superscript indicating the desired value
h	Superscript or subscript indicating a hidden layer
m	Superscript or subscript indicating a model of the given plant
$tr\ (x)$	Trace (spur) of matrix x, $\Sigma_i x_{ij}$
T (or ')	Superscript indicating the transpose of a vector or matrix
$\in$	' is an element of'
$\bar{x}$	Equilibrium value of state vector x

$\hat{x}$	Estimate of x, signified by the use of a caret (hat)
$\lvert x \rvert$	Absolute value (magnitude) of x
$\lVert x \rVert$	Euclidean norm (length) of vector x
$\lVert \cdot \rVert_\infty$	Infinity norm
$\lVert \cdot \rVert_F$	Frobenius norm
σ	Diagonal operator
x^T (or ')	Transpose of vector x, signifies by the superscript T (or " ' " in Matlab)
$\Re$	Real
Δw	Small change applied to weigh w
∇	Gradient operator
Δ	Forward difference
Var	Variance operator
i or j	Square root of -1, (complex value)
H^1	Inverse of matrix H
$\underline{I}$	Identity matrix
Re {x}	Real part of x

ABBREVIATIONS

abs(X)	Matrix with (i, j) element $\lvert x_{ij} \rvert$
arg z	Argument of the complex number z
arg max z (.)	That value of z which maximise (.)
Adaline	Adaptive Linear Element
AI	Artificial intelligence
AR	Autoregressive
ARMA (X)	Auto-Regressive Moving Average (with eXogenous inputs)
ARX	Auto Regressive, eXternal input
ASTE	Averaged Squared Tracking Error
BP	Back-propagation
BPTT	back propagation through time
BSB	brain-state-in-a-box
CMAC	Cerebellar Model Articulation Controller
CAR	Controlled Auto Regressive
deg	Degree (polynomial)
Det	Determinant of matrix; Diag *{x_i}* Diagonal matrix with elements x_1, x_2,... If the matrix is not square, then x_1, x_2,... are the elements on the principal diagonal, and all other elements are zero. The x_i may themselves be matrices

DNMRAC	Direct neural model reference adaptive control
E{x}	Expected (mean) value of stochastic process {x}
e_i	The i^{th} standard basis vector $[0 \ldots 0\ 1\ 0 \ldots 0]^T$, with 1 occurring in the i^{th} position
ELS	Extended Least Squares
FC	Feedback Controller
FEL	Feedback error learning
FIR	Finite Impulse Response
G(A, B, C, D)	Denotes that (A, B, C, D) is a state-space realisation of the transfer function (matrix) G
G(s)	A transfer function (matrix) frequently abbreviated to G
G*(s)	Denotes $G^T(-s)$. (But $G^H(s)$ denotes $G^T(\overline{s}\,)$
gcd{.}	Greatest common divisor of the set {.}
H_∞	Set asymptotically stable transfer functions G, with $\|G\|_\infty < \infty$
GLS	Generalis ed Least Squares
GMVSTC	Generalised minimum variance self-tuning control
GUI	Graphical User Interface
HARMA	Hammerstein Auto Regressive Moving Average
HMFFNN	Hammerstein multi-layer feed-forward neural network
I	Unit matrix of unspecified dimension
IDML	Inverse dynamics model learning
I_n	Unit matrix of dimension n
Im{x}	Imaginary part x
INA	Inverse Nyquist Array.
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
LTR	Loop Transfer Recovery
LTI	Linear Time Invariant
LTV	Linear Time Variant
LHS	Left-hand side
RHS	right-hand side
MFD	Matrix-Fraction Description
ML	Maximum Likelihood
MIMO	Multi-input Multi-output
MLP	Multilayer Perceptron
MRAC	Model Reference Adaptive Control
MLS	Modified Least Square algorithms
MV	Minimum Variance
NMV	Non minimum Variance
NN	Neural Network

NARMA	non-linear auto-regressive moving average
NARX	non-linear auto-regressive with exogenous inputs
NNOE	Neural network output error
OE	Output Error
QLS	Quasi-Least Squares
RLS	Recursive Least Squares
RLMS	Recursive Least Mean Squares
RBF	Radial Basis Function
RFN	Reversed Frame Normalisation
RHONN	Recurrent Higher Order Neural Network
RML	Recursive Maximum Likelihood
r.v.	Random Variable
SG	Stochastic Gradient
SISO	Single input single output
SIMO	Single input multi-output
SSIF	State Space Innovations Form /NNSISF
SPR	Strictly Positive Real
SVD	Singular Value Decomposition
TD	Temporal Difference
TDL	Tapped Delay line
TDNN	Time-Delay Neural Network
TLFN	Time Lagged Feed-Forward Neural Network
VLSI	Very-Large-Scale Integration
w.r.t	with respect to
XOR	exclusive OR

CHAPTER 1: INTRODUCTION

1.1 Introduction

The aim of automatic control is to influence the controlled process in such a way that its behaviour conforms to some desired specification. This is usually expressed in terms of different parameters, for example, temperature in a chemical reactor, speed of a car, global positioning of a satellite (GPS) or the financial cost of running a plant. Automatic control has developed in the post war period into a scientific discipline in its own right and at present control systems are applied and are indispensable, in a wide range of industries.

Further developments in the field are fuelled on the one hand by increasing demands on performance of control systems, expressed in terms of accuracy, speed, cost, reliability, efficiency and so on. On the other hand, technological advances in other areas result in the emergence of new, complex and challenging systems that need to be controlled, for example high-performance aircraft, robot manipulators or advanced underwater and space exploration vehicles and so on.

The so-called classical control theory has been developed for dynamic systems that are described by transfer functions or in state space form, in continuous or discrete time that are linear. Linear control theory is a mature area offering effective and well understood techniques and tools for both analysis and design of controllers for linear systems whose dynamic models are known.

However, most physical systems are to a greater or lesser extent non-linear in nature. The traditional approach relies on designing the control system based on a local linearised model of a non-linear plant around a given operating point. For some systems, whose non-linearities are not too severe and the system's operating point is confined to a small region, such an approach may give good results. However, for many dynamical systems, which are significantly non-linear in their operating region, such an approach is not suitable and linear controllers would yield poor performance or even result in instability. These difficulties are often compounded by uncertainty regarding process parameters or even the structure of the model, also noise, unavailability of measurements of some of the variables of the controlled process and variation of the process characteristics with time.

1.2 Self-tuning Adaptive and Non-linear Control

In response to these challenges and thanks to the ever increasing availability of cheap and rapidly moving computing technology enabling implementation of more sophisticated algorithms, non-linear and adaptive control has recently become an area of strong research activity.

The non-linear control laws took explicit account of the known nonlinearities in plant dynamics. One of the first was the scheme proposed in [Blaschke, 1972], which made use of a non-linear, physically motivated, transformation of state co-ordinates to render the system more simple for the eventual controller design. Subsequent development of differential-geometric theory of non-linear systems, of which good treatment was given in [Isidori, 1989] , put many early results in a unified framework. The so-called linearization by feedback became a very popular non-linear control technique. It consists of a non-linear co-ordinate transformation and non-linear feedback control, which make the closed loop system linear.

The biologically inspired concept of adaptivity entered the picture and stimulated interest in the development of control systems that could adapt to changing or unknown characteristics of the plant. Although the concept appears intuitively understandable, it is quite rich in meaning and it is surprisingly difficult to devise a single precise definition of adaptivity in the systems theory context. An interesting introduction can be found in [Narendra and Annaswamy, 1989]. In the context of the work presented in this thesis, it is suitable to describe an adaptive controller as such which adapts the model of the plant used for control generation or directly the control law based on the measurement information supplied to it. The overwhelming majority of existing adaptive designs are formulated as parameter adaptive systems (as opposed to structurally adaptive systems). That means that a structure of the system model or the control law is fixed and only its parameters are updated. Self-tuning adaptive control was, at least partially, motivated by non-linear and time-varying systems. It is often applied to such systems; design of probably stable adaptive controllers even for linear systems with constant parameters was a major hurdle for the control community.

Two different cases need to be distinguished here. The situation, when the state of the plant is accessible, is a relatively simple one as the control law utilises state feedback and the output feedback case, when full state vector of the plant is not available. This is significantly more difficult since the adaptive controller needs to include a control law which is dynamic (even for constant parameter values). The solutions to this linear adaptive control problem were finally obtained in the early 80's and these results are summarised in a number of books [Narendra and Annaswamy, 1989; Astrom and Wittenmark, 1989] and others.

Although a classical adaptive controller is linear for constant values of its parameters, parameter adaptation makes it altogether a non-linear system. Contrary to early expectations, taking a linear control law, say pole placement and combining it with an estimation module, containing for example Recursive Least Squares (RLS), parameter adaptation of the controller is not enough to guarantee the closed loop stability. In the output feedback case, it turns out that stable adaptation required using specially parameterised, non-minimal in terms of state dimension, models of linear plants.

Very recently, adaptive control of non-linear systems has been more actively pursued, but despite some impressive results, [Marine and Tomei, 1995; Krstic *et al.*, 1995], the existing systematic techniques are limited to some specific classes of systems. Particularly the non-linear adaptive output feedback control is still very much an open problem.

1.3 Neural Networks

The field of artificial neural networks comprises a vast body of research and practical applications in a wide range of disciplines. Although origins of artificial neural networks go back to 1940's, only in the last decade or so has the control community taken a strong interest in the concepts and techniques associated with them. Neural networks in control became for some time a very active topic which was demonstrated by a profusion of papers, journals, conferences and books devoted to this subject. As it is usually the case with fashionable topics, inevitably, some of the proposed applications of neural networks for control were, not based on sound scientific analysis and were often supported by doubtful anthropomorphic arguments. It is however, unquestionable that amongst the wide variety of proposed neural control solutions, there were valuable new concepts, which genuinely benefited the field of automatic control. Gradually, as the initial excitement was fading away, papers with more rigorous approaches to neural control started to appear.

The most important feature of neural networks, from the control applications point of view, appears to be their ability to approximate static non-linear mappings and also provide models for dynamic systems. Their ability to "learn" from data patterns, as the neural network's literature usually puts it, is directly related to adaptive systems. Neural networks also exhibit some advantageous features that are related to implementation issues and which stem from their massively parallel structure.

1.4 Objectives of this thesis

As discussed earlier in Section 1.2, non-linear self-tuning adaptive control remains one of the open challenges. One of the central problems in the adaptive design is that of finding an appropriate representation of the controlled system for the purposes of control synthesis. Such application puts different demands on the representation used compared with pure modelling purposes, i.e., replicating the system's behaviour with the model. This point is very well illustrated by the solution to the linear adaptive control problem, which necessitated special parameterisations of linear plants. Due to the properties of neural networks mentioned above, their main promise for control systems appears to be in the area of non-linear adaptive control.

The aim of this work is to investigate what neural networks can offer for non-linear self-tuning adaptive control. The main interest is in their applicability for representation of non-linear systems in adaptive controller design. The approach taken was to adopt the systems perspective and treat neural networks as a tool for control systems.

It was mentioned in Section 1.2, with reference to linear adaptive control, that the two cases of state accessibility or inaccessibility are significantly different from the controller design viewpoint with the latter being more difficult. The same holds for non-linear adaptive control and in particular for control with neural networks. While in the case of state being accessible, where it is sufficient to use static networks, significant progress has already been made. In the output feedback case relatively few results have been published in the literature. In this thesis, these two cases are treated separately with the emphasis being put on the latter.

This study makes the following contributions:

- non-linear system identification and self-tuning adaptive control using neural network which is based on incorporation of different methodology, such as identification, predictive and self-tuning adaptive control (Chapter 4).
- adaptive control and neural network enhanced generalised self-tuning control (Chapter 5 and 6)
- an overview of neural network (Chapter 2)
- neural network control strategies (Chapter 3)
- Recurrent neural network (Chapter 7)

Most of the results presented in this thesis have been or will soon be published. The following papers have been published and the abstracts are included in appendix C.

B.K.Thapa and T.Earthrowl-Gould (1999) "Back-propagation Neural Networks Enhanced Non-linear System Identification and Control". IEE Sri Lanka Centre 5th Annual Conference' Sep.1999. (Refereed and re-submitted on 18 Aug.2000 for publication for proceeding of IEE Sri-Lanka Centre). To be published

B.K.Thapa, B.Jones and Q.M.Zhu (2000). Nonlinear control with Neural Networks. KES'2000, Fourth International Conference on Knowledge-based Intelligent Information Engineering Systems & Allied Technologies. Proceeding Vol.2 pp.868-873.

1.5 Outline of this Thesis

Chapter 1: This chapter introduces the aims, objectives and contribution of the thesis by highlighting the research background.

Chapter 2: A selective but necessary overview of the wide field of neural networks is presented here. This touches upon the applicability of concepts and techniques for control systems and in particular, their relevance to the adaptive control algorithms discussed in the following chapters. Two neural network structures, feed-forward and recurrent networks, are also discussed.

Chapter 3: This chapter discusses the neural networks control strategies quite intensively. An approach of classifying neural networks control strategies is presented. The most commonly used two fold classifications of neural networks are discussed. Some new control structures are also discussed.

Chapter 4: In this chapter neural network based system identification for linear and non-linear control system application is proposed. The main attention is drawn to:

- Identification of neural network models for non-linear dynamic systems.
- the network architectures, includes static and dynamic, single and mulit-layer and recurrent type of networks.
- various types of neural network learning such as, Hebbian, Perceptron, Delta, Widrow Hoff rule and others.
- Special attention has been paid in this chapter is to develop most efficient learning algorithm for the multilayer neural network; namely, BP learning in dynamic network.
- Non-linear self-tuning adaptive control

Simulation results are presented to demonstrate the algorithm mentioned in the chapter.
Some part of this chapter has been already published in the conference and proceedings (see section 1.4 above for details).

Chapter 5: In this chapter, various types of linear adaptive and non-adaptive control schemes and results are presented so that the techniques described here can be linked directly or indirectly to earlier chapters especially Chapter 4 and Chapter 6

Chapter 6: This chapter present a neural network enhanced self-tuning controller, which is designed by amalgamating neural network mapping with a generalised minimum variance self-tuning control (GMVSTC) strategy. Using this technique, the controller can deal with non-linear plant which exhibits many feature such as uncertainties, non-minimum phase, coupling effects, unmodelled dynamics (assumed to be globally bounded). The unknown non-linear plants to be controlled are approximated by an equivalent model, which is composed of simple linear plus non-linear sub-models respectively. Simulation results are presented to illustrate the algorithms described in the chapter.

Chapter 7: Recurrent networks are neural networks with one or more feedback loops. The feedback can be of a *local* or *global* type. The objective of this chapter is to study recurrent networks with global feedback.

Chapter 8: This chapter highlights the benefits of neural networks and their application for controls as well as other scientific fields.

Chapter 9: This thesis concludes with a summary and discussion of some noteworthy points concerning possible future research directions.

CHAPTER 2: OVERVIEW OF NEURAL NETWORKS FOR CONTROL SYSTEMS

2.1 Introduction

In this chapter, an overview of techniques and ideas from the field of neural networks is presented. This field has been an area of tremendous activity in the recent years and by now comprises a vast body of research and practical applications in many various disciplines ranging from finance to aerospace. It is not the intention here to provide an exhaustive or even representative survey of this wide field, nor even a full survey of numerous applications of neural networks in control. Rather some general aspects of neural networks, which are relevant from the point of view of control systems, are discussed here with the particular emphasis on the issues, which are directly related to the main topics addressed in this thesis.

The two neural structures, which are of particular relevance to this thesis, are the static feed-forward neural networks (FFNN) and the continuous and discrete time recurrent neural networks (RNN). A good general introduction to neural networks is given for example in a book [Haykin, 1994 and 1999] respectively. A broad surveys of neural networks applications for control and identification are presented in papers [Hunt *et al.*, 1992; Narendra & Parthasarathy, 1990; Sjoberg *et al.*, 1995] and in the books by [Miller *et al.*, 1990; Warwick *et al.*, 1992 & 95].

2.1.1 Motivations behind Artificial Neural Networks

As it is immediately suggested by their name, the artificial neural networks are biologically inspired. Observations that humans and animals are much better than conventional computers in performing certain functions, for example image recognition or perhaps more importantly their ability to learn from experience, led to attempts to understand the ways in which information is processed by living organisms.

In recent years, an ever-intensifying effort has been made to study mechanisms and structures of the brain. Origins of artificial neural networks go back to the early 1940s when the first structures consisting of interconnection of elements based on a model of a neurone were investigated [McCulloch and Pitts, 1943].

There are two possible extremes, points of view on what are artificial neural networks and why they could be useful. One approach is to treat them as structures, which emulate behaviour of biological nervous systems. Thus, on the one hand they could help understand functioning of nervous systems in humans and animals and on the other, their applications in practical systems would exhibit certain attributes of biological systems. This type of thinking was visible in some, especially early, literature dealing with applications of neural nets in control systems. Because a control system utilises neural networks, it would therefore, by definition be able to learn from experience, generalise from examples and so on. The other position is to view artificial neural networks simply as a class of mathematical algorithms, which are able to produce solutions to a number of specific problems. The network structure is just a diagrammatic representation of these algorithms.

Present day knowledge about functioning of the brain and the nervous system is still very limited and one should be very careful with drawing analogies between biological and artificial neural systems. For example, it seems rather unlikely that gradient-descent based learning, which is used in most artificial neural networks, is at least one mechanism underlying learning capabilities exhibited by living organisms.

2.1.2 Applicability to Control Systems

To find out what neural networks can offer to control engineering, it is necessary to compare them with the existing established techniques and tools for control systems design. Such comparisons and in general usefulness of neural networks for control have to be judged against criteria applied normally to control systems, e.g., tracking performance, stability, robustness etc., leaving aside (uncertain) "biological" arguments. There are a number of good reasons why neural networks can and are useful, for control systems design. The relevant properties are as follows:

(i) **Inherent non-linearity (or non-linear systems):** Neural networks have a proven ability to approximate arbitrary non-linear mappings, that is, they are capable of modelling of both static and of dynamic systems. Their greatest potential is by far in the area of non-linear control problems. This issue is investigated in this thesis.

(ii) **Learning capability (or adaptive systems):** Neural networks have powerful learning capability and they learn from examples. A network trained on data recorded from a system under study should produce appropriate behaviour when presented with inputs not appearing in the training data. Such training can be performed on-line. From the control systems viewpoint, such properties are directly related to the issues of systems modelling, identification and adaptive control.

(iii) **Capability of generalization:** Most neural networks exhibit some structural capability for generalization. In particular, the network will cover many more situations than the examples used to train it. Therefore, they have the ability to deal with difficulties arising from uncertainty, impression and noise, in a wide rage of problems [Brown *et al.,* 1992].

(iv) **Massive parallelism:** Neural networks have parallel structure, which allows for parallel distributed implementations. The basic processing element has a very simple structure which, when combined with parallel processing technology implementations, can lead to very fast processing.

(v) **Hardware implementation:** Due to parallelism and simplicity of a single neurone, neural network algorithms can be implemented in hardware, bringing the benefit of speed and possible large size of networks used. A number of VLSI hardware implementations are already available on the market [Ramacher and Ruckert 1991].

(vi) **Fault tolerance**: Practical neural systems potentially possess a higher degree of fault tolerance than conventional systems due again to their parallelism. A loss of a single neural cell should not have a significant adverse effect on performance. Moreover, if the system is learning on-line or in other words is adaptive, such loss can be accommodated by appropriate adaptation of other neurones. This could that provide for a property of so-called graceful degradation, which is very much sought after, especially in safety critical control systems.

(vii) **Data fusion:** Neural networks can process both quantitative and qualitative data.

(viii) **Guaranteed stability:** Recent theoretical results prove that the certain neural network control structures are guaranteed to be stable for certain non-linear control problems [Brown *et al.*, 1992].

These above mentioned properties of neural networks, which are related to the implementation issues, provide strong arguments for their use in practical control systems and provide motivation for research into the neural network based control algorithms. In this thesis algorithm implementation issues are addressed.

As far as control algorithm design is concerned, it is certainly the applicability of neural networks for non-linear and adaptive control, which is of greatest importance. Especially, during the early enthusiasm for applying neural networks to all possible problems, there were a number of papers proposing control algorithms based on linear neural models. With the powerful techniques offered by linear control theory and a number of good linear models traditionally used (e.g. ARX, ARMAX, ARIMAX etc.) there seems nothing new, that neural networks can offer for parameterisation of linear dynamic systems. On the contrary, such models can lead to unnecessary over-parameterisation and poor controller performance. This problem is nicely exposed by [Warwick, 1994].

Control of linear time-invariant plants with unknown parameters and un-measurable states has been an area of intense effort for the past three decades but now have reached a stage of relative maturity. In the early 1980s, adaptive algorithms guaranteeing stable control of such systems have been discovered. The choice of a structure of a model of a controlled system, a crucial element of an adaptive controller, was based on results from the theory of linear systems. Well-known stability results for linear systems provided guidance in the choice of adaptive laws governing adjustment of controller parameters. This topic is very well covered in [Narendra and Annaswamy, 1989; Astrom and Wittenmark, 1989; Sastry and Bodson 1989].

In contrast to this, there is no universal theory of non-linear control and reliable techniques exist yet only for specific categories of non-linear systems with many non-linear control problems remaining unsolved. This is especially the case when the dynamics of non-linear systems are inadequately known or are time varying and state variables are not accessible. This is a situation, which requires some kind of adaptive controller.

Similarly, as in the case of adaptive control of linear systems, a central issue in the suitable representation of a non-linear system, is the synthesis of an adaptive controller for non-linear plants. Of course, due to a dramatically bigger variety of possible behaviour in non-linear systems compared to linear ones, it is unlikely that a universal adaptive non-linear controller can be found. Still, universal approximation property of neural networks makes them potentially applicable to various non-linear adaptive control problems. The obvious question is why neural networks should be better than other more conventional non-linear approximation schemes. Without attempting to provide an exhaustive answer or a comprehensive comparison, the favourable properties of neural networks for use in non-linear self-tuning adaptive control will be highlighted in this thesis.

2.2 Network Architectures

A neural network is defined by the structure of its basic elements, the way these basic elements are interconnected and by the learning rules.

2.2.1 Neurones

The basic processing element of connectionist architecture is usually called a neurone by rather loose analogy with biological systems. A general model of a single neurone is presented in Fig. 2.1. It consists of a weighted summer, a linear dynamic SISO system and a static, usually non-linear, function also called the activation function and is selected differently in different applications.

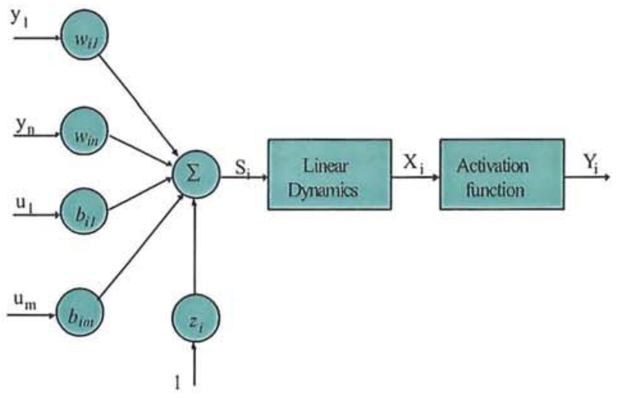


Fig.2.1 Model of a neurone

2.2.2 Weighted summer

The weighted summation can be described by

$$s_i(t) = \sum_{j=1}^{N_i} w_{ij} y_j(t) + \sum_{k=1}^{M_i} b_{ik} u_k(t) + z_i$$

Thus signal s_i consists of a weighted sum or net outputs y_i of all neurones and external inputs u_k and of a bias term z_i. Naturally, the above equation can be represented for the whole network in a vector-matrix notation as:

$$s = Wy + Bu + z$$

where y and z are vectors, of dimension n, of outputs from neurones and biases respectively. u is the external input vector and ***W*, *B*** are weight matrices of appropriate functions.

2.2.3 Linear Dynamics

The linear dynamic system with input s_i and output z_i can be either continuous or discrete-time and in general can be described by either differential or difference equations respectively, or by appropriate transfer functions. Usually, the system dynamics are quite simple (i.e. low order). A simple inertia is often used, which in continuous time is described by

$$T_i \dot{x}_i + x_i = s_i$$

A discrete-time version can be described by

$$\alpha_1 x_i(k+1) + \alpha_2 x_i(k) = s_i(k)$$

where k is an integer time index. A trivial version of the dynamics is a system where the relationship between input and output is static

$$x_i = s_i$$

This is the case in static neural networks.

Clearly, choice of the dynamics in a neurone is unlimited. Particular application of a neural system can provide guidance here and there is a lot of space for heuristics. For example, if the network is used for modelling of a plant and the plant is known to contain time delays, delays can be incorporated into the dynamics of neurones.

2.2.4 Activation Function

The activation function describes a static relationship between the output y_i of a neurone and the output of its dynamic part x_i.

$$y_i = f(x_i)$$

A number of most common functions used are as shown in Fig. 2.2 (a-c). Two important classifications of these functions are:

- Differentiable versus non-differentiable functions

- local versus global functions

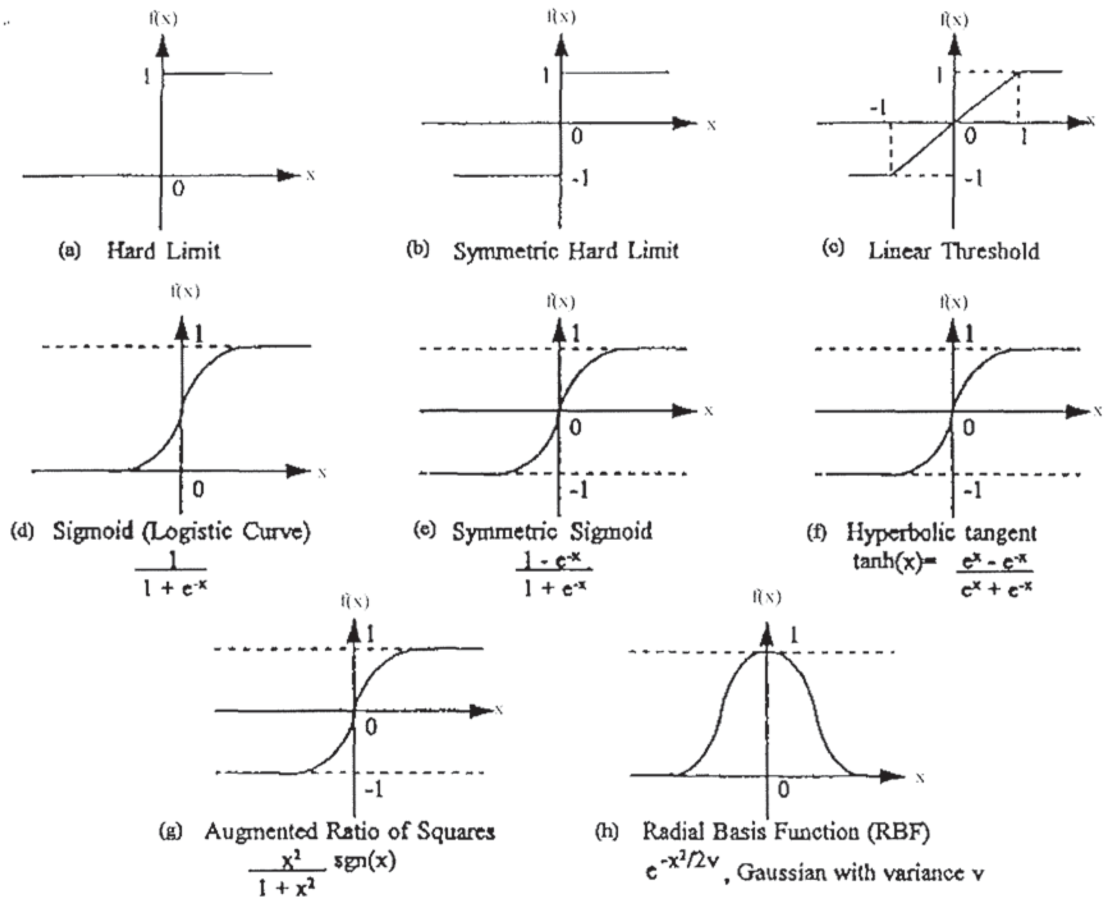


Fig. 2.2 Activation Functions: Threshold (a, b, c), Sigmoid (d, e, f, g) and Gaussian (h).

The differentiability of the activation function is needed for all weight adaptation algorithms, which utilise gradient such as back-propagation. Step functions, which are examples of non-differentiable activation functions are shown in Fig. 2.2 (a, b). Such functions are used in networks, which need to produce a binary output. Algorithms discussed in this thesis employ only networks with differentiable activation.

The second classification distinguishes between activation functions, which have significant output only in a small region of input space and those support covers a large part of the input space.

A differentiable function $\sigma(x)$ is called a sigmoid function e.g. [Sjojberg *et al.*, 1995] if it has the property that both limits exist

$$\lim_{x \to -\infty} \sigma(x) = a$$
$$\lim_{x \to +\infty} \sigma(x) = b$$

and are distinct. Without loss of generality it is usually assumed $a < b$. Thus, sigmoid functions have global support. Two common choices of sigmoid functions (shown in Fig. 2.2 (d-g)) are:

$$\sigma_1(x) = \frac{1}{1+e^{-x}}; \sigma_2(x) = \frac{1-e^{-x}}{1+e^{-x}}; \sigma_3(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \sigma_4(x) = \frac{x^2}{1+x^2}\operatorname{sgn}(x)$$

Gaussian functions as shown in Fig. 2.2 (h).

$$y = e^{-x^2/v^2} \text{ or } y = e^{-x^2/2v}$$

is an example of differentiable activation, which has significant output only locally.

2.2.5 Connections

Although single neurones themselves are very simple, an interconnection of a large number of them allows obtaining powerful computational capabilities and very rich behaviour. Regarding the choice of a structure of single neurones, there are many ways, neurones can be interconnected and the choice is normally dependent upon the application of a neural network.

For example, a common structure is one with all neurones non-dynamic. In such a network all relationships can be described by a set of algebraic equations

$$\begin{aligned} x &= Wy + Bu + z \\ y &= f(x) \end{aligned} \tag{2.1}$$

where *f (X)* is a vector whose components are *f (x$_i$)*. The network has no memory and the output of its neurones depends only on the instantaneous value of the input vector ***u***, weight matrices ***W***, ***B*** and the bias vector ***z***. Static networks whose neurones are arranged in layers are by far the most common type in control and identification applications and will be treated in more detail in a separate section.

If some or all the neurones contain dynamics, the neural network becomes a non-linear dynamic system. If dynamics are continuous-time, the network can in general be described by a set of differential equations

$$\begin{aligned} \dot{x} &= g(x, u, \Theta) \\ y &= h(x, \Theta) \end{aligned}$$

where x is the state vector and vector contains weights of the network. The interconnection structure of the network is encoded in the function g and relations between its state and output vector are described by the function h. Discrete-time dynamic networks can be described by a corresponding set of non-linear difference equations. Again, a separate section is devoted to a certain class of dynamic (recurrent) neural networks; namely, the Hopfield type networks, since their application to control is the main issue investigated in this thesis.

2.2.6 Learning Rules

Once the basic structure of neurones and their interconnections are decided and thus the network structure is fixed, a network exhibits fixed behaviour for constant values of weights. The learning process of the network is then concerned with determining such values of the weights so that the network behaviour is closest to the desired one. This can be for example correct classification of input patterns, approximation of a non-linear function or convergence to a particular equilibrium point in case of a dynamic network.

In the context of learning in static networks, the connectionist literature distinguishes them into two main classes of learning algorithms: the so-called ***supervised*** and ***unsupervised*** learning.

In the supervised learning a desired response $\boldsymbol{d}$ is known for every input applied. Thus, an external reference signal (teacher) is available. The distance between the actual network output y and the desired one $\sigma(d,y)$, providing a measure of the error, is computed and used to adjust network weights. A training set consisting of input and output patterns is required in this training mode.

In the unsupervised learning, only very little *a priori* information is available - the desired network response is not known. Thus, explicit information about output error cannot be used to adapt the weights and learning can only rely on local information and internal signals.

It is mostly the supervised methods that are used for neural networks in control applications. Algorithms described in this thesis employ only methods falling into this category and therefore unsupervised learning will not be discussed here.

What in Artificial Intelligence and connectionist circles is understood by learning falls exactly into the framework of parameter estimation (adaptation), one of the central issues in the control and identification theory and practice. For example, batch training using a set of input and output measurements of the plant, aimed to obtain a network providing a good model of the plant, is nothing else but off-line identification. The so-called on-line learning corresponds in adaptive control to on-line parameter adaptation. Particularly learning rules or in other words parameter estimation or adaptation techniques will be discussed in the following sections in the context of specific network architectures: feed-forward and recurrent.

2.3 Static Multi-layer Feed-forward Neural Networks (MFFNN)

A particularly popular and useful in control and identification applications is a static structure with neurones arranged in layers. Such structure has illustrated in Fig.2.3. The name feed-forward (or static) refers to the one-way direction of how in computations when the network output is calculated. This is illustrated by arrows in Fig.2.3 A feed-forward network with identical sigmoidal activation functions in the nodes of its hidden layers usually called a *Multi-layer Perceptron* (MLP).

2.3.1 **Multi-layer Perceptron** (MLP)

In a MLP, the point of entry for external input is called an input layer. Input layer nodes pass the inputs unmodified. Inputs $x_1,\dots,x_n$, are then multiplied by the first set of weights (contained in the matrix W^1) and these weighted sums of inputs, together with biases of the first layer, form inputs to the sigmoid activation functions of the first hidden layer.

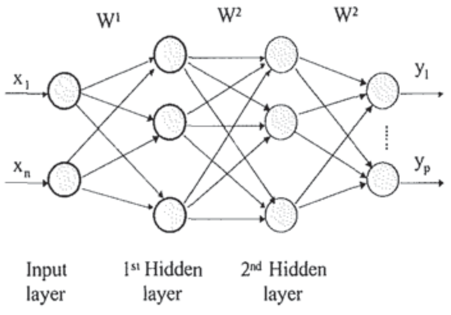


Fig. 2.3: General FFNN with two hidden layers.

Thus, in a vector-matrix notation, vector y^1 containing outputs of the first hidden layer is given by

$$y^1 = \sigma(W^1 x + z^1)$$

where $x = [x_1,\dots, x_n]^T$ is the input vector and z^1 is a vector of the biases of the first hidden layer, whose size is equal to the number of nodes in this layer. The non-linear operator $\sigma(x) = [\sigma(x_1),\dots, \sigma(x_n)]^T$. In a similar fashion, outputs of the first hidden layer are fed into the second hidden layer and so on, (if there are more than two hidden-layers). The nodes of the output layer contain linear activation $(f(x_i) = x_i)$ and usually no biases and thus outputs of the whole network are weighted sums of outputs of the last hidden layer. Naturally, the whole network can be described by equations of the form (2.1) with an appropriate construction of matrices ***W*** and ***B***. It is however much more revealing to take explicit account of the layered structure when describing mathematically such networks. With slight abuse of notation, the bias vectors can be included in the weight matrices by augmenting inputs to a layer with l's. Then, the p-dimensional output vector y of the network in Fig.2.3 is given by

$$y = W^3 \sigma(W^2 \sigma(W^1 x))$$

A fundamental property of the multi-layer perceptron is its ability to approximate arbitrary non-linear mappings.

2.3.2 MLP as a Universal Approximator

The following theorem in [Funahashi, 1989] establishes ability of a network with a single hidden layer to approximate, arbitrarily well, continuous functions over compact sets:

Theorem 1 [Funahashi, 1989]: Let $\sigma(x)$ be continuos, non-constant, bounded and monotonically increasing function on **R** (i.e., sigmoid and monotone). Let K be a compact subset of $\boldsymbol{R}^n$ and $f(x_1,...,x_n)$ be a continuous real valued function on K. Then, for an arbitrary $\in > 0$, there exists an integer N, real constants $w_i^2, z_i (i = 1,..., N)$ and $w_{ij}^1 (i = 1,...,N; j = 1,...,n)$ such that

$$\hat{f}(x_1,...x_n) = \sum_{i=1}^{N} w_i^2 \sigma(\sum_{j=1}^{n} w_i^2 x_j - z_i) \tag{2.2}$$

satisfies

$$\max_{x \in K} \left| f(x_1,..., x_n) - \hat{f}(x_1,..., x_n) \right| < \in$$

Of course, formula (2.2) describes nothing else but a single hidden layer network. Similar results were obtained by [Cybenko, 1989; Hornik *et al.*, 1989]. The above theorem immediately extends to networks with more than one output. A similar property for networks with more than one hidden layer can be derived from the above theorem or be shown from scratch [Funahashi, 1989]. The requirements on the activation function are relaxed to continuous, non-constant and bounded functions in [Hornik, 1991].

In [Leshno *et al.*, 1993] a universal approximation property is shown for all non-polynomial activation functions which are locally Riemann integrable (continuity is not required). However, smoothness of the activation function has the crucial advantage that the derivatives of network outputs with respect to weights can be computed and thus gradient based parameter estimation methods can be applied.

The above theorem is an existence result and does not specify the required number N of hidden layer nodes. There is no prescriptive method to choose an appropriate network size for a given approximation problem and one has to apply a heuristic method. Although, one hidden layer is enough for a universal approximation of continuous functions, it appears that use of more hidden layers might result in a more *"efficient"* approximation scheme, that is, to achieve the same quality of approximation a smaller total number of hidden nodes is required. This relation was observed during the simulation studies with static networks reported in this thesis and similar observations were made for example in [Chen, 1990; Hunt *et al.*, 1992; Sjoberg *et al.*, 1994]. Also, it is mentioned in [Sontag, 1993] that in some control applications, where discontinuous control law is required, a two hidden layer net can stabilise a system which cannot possibly be stabilised with a single hidden layer network.

2.3.3 Relation to Other Non-linear Approximation Schemes

The universal approximation property of FFNNs is not unique and a natural question is how does this scheme compare with other more traditional approximation techniques like for example polynomials or orthogonal expansions. The general problem of the approximation theory is to find a way of describing a true relationship

$$y = f(x_1,\dots,x_n) = f(x)$$

based on the available set of measurements $\{x_k, y_k\}$. k is the index of the subsequent input-output pairs and the total number of them is, say, S. For control systems purposes, function $f : R \to R^n$ can normally be assumed continuous. Treating, for simplicity, f as a real valued function does not restrict generality and the argument naturally extends to functions whose output space is $\mathbf{R}^p$ rather than $\boldsymbol{R}$.

Sometimes certain information is available about the original function f. For example, a physical insight can suggest what type of functional relations exist between variables and the problem is then reduced to finding a set of parameters like resistance, friction coefficients etc. If such information is not available, general parameterised families of functions are used which are capable of describing any reasonable function. In system control and identification applications, this is called a *"black-box"* approach. Usually, a function expansion of the following form

$$\hat{f}(x,w) = \sum_{i=1}^{N} w_i f_i(x) \qquad (2.3)$$

is used, where f_i's are functions

$$f_i : R_n \to R$$

and $\boldsymbol{w}$ is a parameter vector whose components are w_i's.

A natural requirement is that the set of functions $\{f_i(x)\}$ forms a basis for a class of functions to which f belongs. Usually, this is a set of real valued continuous functions over a compact subset K of R^n, denoted by $f \in C(K)$, with the norm in $\boldsymbol{C}(K)$ defined as

$$\|f\| = \sup_x\{|f(x)| : x \in k\}$$

thus, for a certain set of suitable parameters w_i it should hold that

$$f(x) = \sum_{i=1}^{\infty} w_i f_i(x) \tag{2.4}$$

There are many possible sets of basis functions $\{f_i(x)\}$, like for example polynomials, Fourier series and amongst the more recent schemes fuzzy models see [Wang, 1994]. The choice of a basis for a given application is determined by practical considerations. In system identification the important criteria are following [Sjoberg *et al.*, 1994]:

I. The approximation error

$$\left\| f(x) - \sum_{i=1}^{N} w_i f_i(x) \right\| \tag{2.5}$$

decreases quickly as N increases.

II. There exists a way of excluding *"spurious"* (not genuine) basis functions from the expansion.

In practice, only a finite number of basis functions can be used and thus it is necessary that a good approximation can be achieved with a finite number of basis functions and a finite number of parameters. The smaller number of parameters or the more parsimonious the representation, the better as it limits complexity of computations. Another important reason for both the above requirements, in system identification applications, is due to noise. The set of available data for the function approximation task $\{x, y\}$ usually comes from measurements of some variables of the plant being identified. These measurements are often contaminated with noise. A trade-off exists in the choice of the number of basis functions with respect to the overall error of the approximator estimated from noisy data see [Sjoberg *et al.*, 1994]. It is a trade-off between the inevitable approximation error expressed by (2.5), the so-called *bias* error and *a variance* error. This variance error increases with increasing number of estimated parameters. Thus, if there are too many parameters in the approximator, some of them contribute very little or nothing at all to the achievable quality of approximation

(2.5) while increasing the variance error. Therefore, there is a need for a method, which allows striking a good compromise and choosing an *"optimal"* number of basis functions.

2.3.4 FFNN as a Function Expansion

Comparing equation (2.2) and (2.3), it is apparent that a single hidden layer network can be interpreted as a function expansion of the form (2.3). Arranging hidden layer weights w_j^1 in vector $w_j^1 = [w_j^1, \ldots, w_{jn}^1]$

$$\hat{f}(x, w) = \sum w_i^2 \sigma(w_i^1 x - z_i) \qquad (2.6)$$

where **w** is a vector containing all weights of the network.

The basis functions are of the form $f_i(x) = \sigma(w_i^1 x - z_i)$. They are obtained by scaling and translating one fundamental function of single variable $\sigma(.)$. This way of looking at neural network based approximators is pursued in [Sjoberg *et al.*, 1994] and some useful properties are pointed out.

Neural networks are quite *"efficient"* expansions in the sense of requirement **I** for functions whose non-linearities are localised, i.e., there is less asymptotic behaviour. Such functions are common in most physical systems. This property is more precisely illustrated by the following result from [Barren, 1993]. Consider a class of function *{f}* on $\boldsymbol{R}^n$ for, which there is a Fourier representation $\boldsymbol{F}$ that satisfies $C_f = \int |\omega| |F(\omega)| d\omega < \infty$

Then, there is a linear combination of sigmoidal function (2.6) such that the following holds

$$\int_{B_r} f(x) - \hat{f}(x) dx \leq \frac{(2rC_f)^2}{N}$$

where $\boldsymbol{B}_r$ is a ball with radius r. The important characteristic of the above bound on the approximation error is that it is dependant on the number N of basis functions but it does not depend on the dimension n, of the input space. A known problem in the non-linear approximation is the so-called *"curse of dimensionality"*. With the growing dimension n of the input space, for a construction of a general non-linear approximator the number of parameters needed and the number of data needed to achieve good parameter estimation grows typically in the power of n. Thus, the problem becomes practically not tractable for larger input dimensions. However, many functions exhibit in some sense a low-dimensional character, that is, their significant behaviour occurs only in localised regions of the input space, or data patterns *{x,y}* are clustered in subspaces. Such property can be exploited in the construction of the approximator.

The idea behind adaptive methods in approximation is to use the data also to select the basis functions rather than work with a rigid basis and only estimate the parameters weighting the summation of basis functions. This approach is for example used in the projection pursuit regression [Friedman and Stuetzel, 1981] and wavelets theory.

The neural network expansion (2.6) uses $\{\sigma(w_i^1 x - z_i)\}$ as its basis functions. These radial basis functions (RBFs) are in fact selected based on data because hidden layer parameters w_i^1, z_i are estimated from data. Therefore, neural networks can be interpreted as function expansions where basis functions are chosen adaptively and which provide an explanation for their good approximation properties.

2.3.5 Redundant Parameters and Over-training

It is shown in [Sjoberg, 1995] that there is a good way of handling requirement **II** in neural network models is by explicit or implicit regularisation. Typically in model identification, whether using neural networks or other schemes, the parameter estimates are obtained by minimising with respect to the parameter vector ***w*** a sum of squared errors over the available data set

$$E = \sum_{k=1}^{S} (y_k - \hat{f}(x_k, w))^2$$

One of the traditional methods, used in system identification and statistics to deal with the problems caused by excessive number of parameters, is to modify the above criterion by adding a so-called regularisation term

$$\overline{E} = E + \delta(w - w^*)^2$$

and minimise such criterion with respect to **w.w*** is usually based on some guess where the correct values of the parameters are located and δ is a positive constant. Regularisation typically reduces the variance error of the approximator but increases the bias error, as in the minimisation process parameter estimates are *"pulled"* towards $\mathbf{w}^*$ which is usually different from the optimal parameter vector.

A well-known problem described in the neural network literature is that of *over-training* or *over-learning*. In the learning process when gradient based minimisation techniques are used, it can happen that with consecutive iterations, while network is improving a fit to the data set on which it is trained, the generalisation between the data points is worsening. Thus, at some stage a further decrease of the error criterion used for training actually results in a poorer quality of the overall approximation of the desired mapping. A good analysis of this phenomenon is given in [Slotine and Sanner, 1993]. Some heuristic techniques have been proposed to overcome this problem. One is to stop the minimisation procedure before the minimum has been found. One way of implementing this is to introduce a so-called *"dead-zone"*. In this scheme the weights updates are not performed if the error falls below a certain threshold. The idea behind it is that if the overall approximation error cannot be reduced below a certain inevitable residue value, further weights updates will not improve the fit anymore but can only make it worse. Another technique used not only for neural networks but also generally in modelling is to split the available-data record into a training set and a validation set. The minimisation is performed using the training set data, while monitoring the error for the validation set. After initial decrease of both training and validation error criteria; at some point the error over the validation set starts to increase. This is usually a signal that over-training has begun and minimisation can be stopped at this point.

It is shown in [Sjoberg, 1995], that there is a connection between these two approaches, in the sense, that stopping the iterations before the minimum is found gives similar results to regularisation and thus can be interpreted as implicit regularisation. It has to be pointed out here that the regularisation techniques discussed above do not find direct application in adaptive control. This is because in adaptive control, parameter adaptation techniques, although also based on reducing some measure of error, are usually designed primarily with the objective of achieving stability of the overall system. However, there exist strong connections between the over-training phenomenon and adaptive control with neural networks. It is shown in [Slotine and Sanner, 1993] that the underlying mechanism behind over-training is a growth of some of the network weights in subsequent iterations. This growth can possibly be unbounded. This problem is known in *adaptive control* as the parameter *drift instability*, described for example in [Narendra and Annaswamy, 1989]. Due to disturbances or even small discrepancies between the controlled plant and the model used by the adaptive controller, adaptation of some of the parameters can lead to their unbounded growth. A number of preventive measures have been developed, use of a dead-zone being one of them.

2.3.6 Some Other Useful Properties of Neural Approximations

The properties of static neural networks discussed so far have been related to general approximation issues, that is, of obtaining a good model of some mapping describing a relationship between a number of variables. Such formulation applies directly to identification (*on-line* or *off-line*) of dynamic systems. In case of adaptive control design, construction of the model of the controlled plant used by the controller is usually driven not only by its approximation capabilities but also by other requirements, the main objective being to achieve a stability of the overall system. Therefore, certain properties of the approximation scheme, which might not be relevant for pure identification purposes, might be vital for successful adaptive control design. In this context, some further properties of neural approximators are presented here.

Sigmoid activation functions are in a sense a quite *"mild"* type of non-linearity. They are monotone, infinitely smooth and have all their derivatives bounded. Let us consider a continuous function over a compact subset K

of R^n, $f \in C(K)$. Due to Theorem 1, this function can be expressed by an approximation realised with a single hidden layer network (2.2) and a functional reconstruction error $\in (x)$

$$f(x) = w^2 \sigma(w^1 x) + \in (x) \tag{2.7}$$

where $\boldsymbol{W^2}, \boldsymbol{W^1}$ are weight matrices and for clarity of notation bias terms are augmented into $\boldsymbol{W^1}$ and σ is a diagonal operator. Matrices $\boldsymbol{W^2}$, $\boldsymbol{W^1}$ contain the theoretical *"ideal"* weights, that is such which give the best approximation over $\boldsymbol{K}$. In a practical approximation problem they are generally unknown. The actual neural model

$$\hat{f}(x) = \hat{w}^2 \sigma(\hat{w}^1 x) \tag{2.8}$$

contains the estimates of those weights $\hat{w}^2$, $\hat{w}^1$, which are different from W^2, W^1. The error between the neural model and the true function it approximates can be expressed as

$$h(x) = f(x) - \hat{f}(x) = w^2 \sigma(\hat{w}^1 x) - \hat{w}^2 \sigma(\hat{w}^1 x) + \in (x) \tag{2.9}$$

The weighting error can be defined as a difference between the current estimate and the ideal value as

$$\tilde{w}^2 = w^2 - \hat{w}^2; \qquad \tilde{w}^1 = w - \hat{w}^1$$

The error between the function realised by the network using the current weight estimates and the best attainable approximation can be expressed in the following way:

$$\begin{aligned}
&W_2 \sigma(W^1 x) - \hat{W}^2 \sigma(\hat{W}^1 x) \\
&= W^2 \sigma(W^1 x) - \hat{W}^2 \sigma(\hat{W}^1 x) + W^2 \sigma(\hat{W}^1 x) - W^2 \sigma(\hat{W}^1 x) \\
&= \tilde{W}^2 \sigma(\hat{W}^1 x) + W^2 (\sigma(W^1 x) - \sigma(\hat{W}^1 x))
\end{aligned}$$

The Taylor series expansion of $\sigma(W^1 x)$ can be written as:

$$\sigma(W^1 x) = \sigma(\hat{W}^1 x) + \sigma'(\hat{W}^1 x)\tilde{W}^1 x + H(\tilde{W}^1 x)$$

where *H(.)* denotes higher order terms. Using the above

$$\sigma(W^1 x) - \sigma(\hat{W}^1 x) = \sigma'(\hat{W}^1 x)\tilde{W}^1 x + H(\tilde{W}^1 x)$$

The following property can then be shown [Lewis *et al.*, 1993].

Property 1: *For the two sigmoid functions (described in Section 2.2.1) the higher order terms of the Taylor expansion are bounded by*

$$\left\|H(\tilde{W}'x)\right\| \le c_1 + c_2\left\|\tilde{W}^1\right\|_F \left\|x\right\|$$

$\|\|_F$ denotes a Frobenius norm which for $A = [a_{ij}] \in R^{mxn}$ is defined as:

$$\|A\|_F = tr(A^T A) = \sum_{i,j} a_{ij}^2$$

The above property is exploited in [Lewis *et al.*, 1993] in the design of a stable adaptive controller based on a static network. Some further properties along these lines are shown in [Chang *et al.*, 1996] and again are crucial in the design of a stable adaptive law described in this paper.

The above mentioned property is essentially based on the fact that higher order terms of the Taylor series of a sigmoid function can be bounded by a linear function of a norm of its argument. Polynomial models, which are one of the most popular traditional techniques, do not possess such feature as they consist of unbounded, exploding terms.

Another related property is that neural networks are capable of arbitrarily accurate approximation not only of smooth functions but also of their derivatives. This result was shown in [Hornik *et al.*, 1990] and later improved in [Hornik, 1991]. This ability is due to smooth and bounded nature of their activation functions. It implies good interpolation capabilities, that is, modelling of function in between data points, which were used for training. Again, polynomials with their divergent terms provide far worse models in this respect.

2.4 Learning in MLP Networks

Learning is concerned with estimation (or adaptation) of network parameters usually formulated as obtaining the best fit to the available data within a given network structure.

Consider a discrete-time non-linear dynamic system governed by the following relationship

$$y(k) = f(y(k-1), y(k-2), u(k-1))$$

where $y(k)$ is the system output (scalar), $u(k)$ is a scalar input to the system and $f(\cdot)$ is a real-valued non-linear function. k is the integer index of the discrete time instants. Suppose we are trying to model the above system using a neural network. Assuming knowledge of the structure of the original mapping, the input to the network will be constructed as:

$$x(k) = [\hat{y}(k-1), y(k-2), u(k-1)]^T$$

and the network will have a single output. After the choice of the number of hidden layers and the number of neurones has been made, the weights need to be determined. Typically, this would be done by a minimisation of an error criterion

$$E = \frac{1}{2}\sum_{k=1}^{S}(y(k) - \hat{f}(x(k), \hat{w}))^2$$

w.r.t the set of weights of the network (for convenience of notation they are placed in a single vector $\hat{w}$). $\hat{f}(.,\hat{w})$ denotes the mapping realised by the network. S is the length of the training set.

Since $\hat{f}(.)$ is a non-linear function of the estimated parameters $\hat{w}$, no analytic solution is available to solve such problems and minimisation has to be performed numerically using a search procedure. Typically, such procedure updates the estimates iteratively in the direction of decrease of the error criterion using a formula

$$\hat{w}(i) = \hat{w}(i-1) - \lambda P(i) - \frac{\partial E}{\partial \hat{w}}$$

where i is the iteration number. λ is the step size or the learning rate. Matrix $\boldsymbol{P}$ is used to modify the search direction. In the simple case $\boldsymbol{P=I}$ we obtain a pure gradient (steepest descent) algorithm. To improve efficiency, second order methods, which approximate Newton direction from the curvature information accumulated in consecutive gradient calculations, can be used, like for example Gauss-Newton, Levenberg-Maquard or conjugate gradient [Fletcher, 1987].

Identification can be performed in a batch mode *(off-line)* when the whole data set is immediately available and parameter updates are based on the whole data record, or recursively *(on-line)* when data only up to current measurement is available and parameter estimates are updated as the new data is acquired.

A somewhat different situation arises when the network is used in an adaptive controller with weights updated on-line. The design of the parameter update law is also based on reducing some measure of error but has to take into account also the overall stability of the system.

2.4.1 Calculation of the Gradient-Backpropagation (BP)

Whether the parameter update law is directly based on reducing the network modelling error or is constructed with stability of adaptive control scheme in mind, in any case it requires calculation of the gradient of some error measure, involving the network output, w.r.t. the network weights. The BP algorithm is a way of computing this gradient and it can be derived by applying the chain rule of differentiation to the expression describing output of the network as a function of its weights.

As an example, let us consider a network with two hidden layers described by

$$\hat{f}(x(k),w) = \sum_{i=1}^{N_2} w_i^3 \sigma\left(\sum_{j=1}^{N_1} w_{ij}^2 \sigma\left(\sum_{l=1}^{n} w_{jl}^1 x_l \right)\right) \tag{2.10}$$

which for clarity of derivation has only one output. The input to the network is $x=[x_1,\ldots,x_n]^T$. N_1 and N_2 are the numbers of nodes in the 1st and 2nd hidden layer respectively. Weight w_{ij}^k is the weight between output of the jth node of the *(k-l)*th layer and *i*th node in the *k* layer (the 1st hidden layer is given the number $k=1$). Thus, for example, w_{ij}^2 denotes a weight connecting *j*th node of the 1st hidden layer with ith node of the 2nd hidden layer. The bias parameters are not explicitly written. Let us denote inputs to the kth layer as x_i^k and outputs as y_i^k. Thus,

$$y_i^k = \sigma(x_i^k) \text{ and for example, } y_i^2 = \sigma(x_i^2) = \sigma\left(\sum_{j=1}^{N_1} w_{ij}^2 y_j^1 \right)$$

Let us assume a simple error function

$$E = \frac{1}{2}\left(y - \hat{f}(x,w)\right)^2$$

where y is the desired output for the input **x.**

For the weights of the output layer the derivatives are given by:

$$\frac{\partial E}{\partial w_i^3} = \left(\hat{f}(w) - y\right)\frac{\partial \hat{f}(w)}{\partial w_{ij}^2} = \left(\hat{f}(w) - y\right)\frac{\partial \hat{f}(w)\partial y_i^2}{\partial y_i^2 \partial w_{ij}^2} = \left(\hat{f}(w) - y\right)w_i^3 \sigma'(x_i^2) y_j^1$$

by defining:

$$\delta_i^2 = \left(\hat{f}(w) - y\right)w_i^3 \sigma'(x_i^2)$$

we obtain the expression for the derivative

$$\frac{\partial E}{\partial w_{ij}^2} = \delta_I^2 y_j^1 \tag{2.11}$$

The derivative w.r.t. the weights of the 1st hidden layer are give by:

$$\frac{\partial E}{\partial w_{jl}^1} = \left(\hat{f}(w) - y\right)\sum_{i=1}^{N_2} \frac{\partial \hat{f}(w)}{\partial w_i^2} \frac{\partial y_i^2}{\partial w_{iL}^1}$$
$$= \left(\hat{f}(w) - y\right)\sum_{i=1}^{N_2} w_i^3 \sigma'(x_i^2) w_{ij}^2 \sigma'(x_j^1) x_l$$

since

$$\frac{\partial y_i^2}{\partial w_{jl}^1} = \sigma'(x_i^2) w_{ij}^2 \frac{\partial y_j^1}{\partial w_{jl}^1} = \sigma'(x_i^2) w_{ij}^2 \sigma'(x_j^1) x_l$$

Again we have: $\delta_j^i = \sum_{i=1}^{N_2} \delta_i^2 w_{ij}^2 \sigma'(x_j^1)$ and obtain

$$\frac{\partial E}{\partial w_{jl}^1} = \delta_j^1 x_l \tag{2.12}$$

Signals δ_j^k are sometimes called the *equivalent error* signals. From the above derivations it can be noticed, that calculation of the equivalent error signals involves propagation back-wards through the net (as opposed to forward calculation when calculating the network output). That is where the name BP comes from. A useful feature of BP is that signals x_i^k and y_i^k are already computed in the forward pass through the net. Except for δ_j^k signals, which are back propagated, the rest of the derivative calculation involves quantities associated with a particular node. The computations are further simplified by simple expressions for derivatives of the two sigmoid functions

$$\sigma_1(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \sigma_2(x) = \frac{1}{1 + e^{-x}}$$

These derivatives are given by

$$\sigma_1'(x) = 1 - \sigma_1^2(x); \qquad \sigma_2'(x) = \sigma_2(x)(1 - \sigma_2(x));$$

Exploiting the above properties, gradient can be computed in a parallel fashion in a special network adjoin to the original one. Such a structure was proposed in [Narendra and Parthasarathy, 1990].

2.4.2 Local Minima

Because output of an MLP network (2.10) is a non-linear function of the estimated parameters **w**, training of such a network is a non-linear optimisation task and a fundamental problem with those is that the error function may have many local minima.

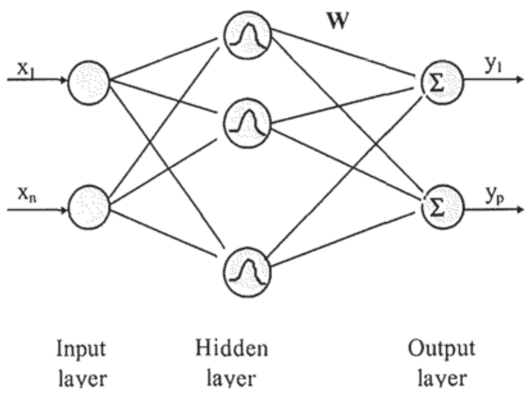


Fig. 2.4: Radial Basis Function network.

Gradient based techniques cannot ensure convergence to the global one. If the training is performed off-line, the situation is easier as a number of methods can be used to try to overcome this problem. For example, the minimisation can be restarted from different initial weights or global strategies like genetic algorithms can be used. In addition, reaching the global minimum is not necessary as some local minima may provide approximation, which is sufficiently good. If the network is used for on-line control, such techniques cannot be applied. Very often off-line pre-training is used to get good initial values of weights. To achieve stability of the controlled system different *"fixes"* can be applied which do not directly address the problem of convergence to a minimum but are explicitly concerned with ensuring stability.

2.4.3 Radial Basis Function Networks

Another variant of a static feed-forward architecture is the Radial Basis Function (RBF) network, which is presented in Fig.2.4. The topology of the RBF network is similar to that of the MLP network but the way inputs are processed is different. It contains a single hidden layer consisting of an array of nodes. Each of these nodes has an associated parameter vector called a centre, whose dimension is equal to the dimension of the input vector. For each of these nodes, an Euclidean distance between the input vector and the centre of the node is calculated. This distance provides an input to a single variable non-linear function contained in each node. Outputs of the whole network are computed as weighted sum of the outputs of the hidden layer nodes. The overall input-output mapping realised by a RBF network is described by

$$y_j = \hat{f}_j(x) = \sum_{I=1}^{N} w_{ji}\phi(\|x-\xi_i\|, v_i), \qquad j=1,\dots,p \tag{2.13}$$

N is the number of nodes in the hidden layer. w_{ji} are the weights of the weighted summation realised in the output layer. They can be arranged into a matrix $W \in R^{pxN}$. ξ_i are the centres of hidden layer nodes. $\|.\|$ denotes an Euclidean norm. $\phi(.,v_i): R^+ \rightarrow R)$ are the non-linear functions of the nodes where v_i are positive scalar parameters determining the "width" of the function.

Some possible choices of the node functions are:

- a multi-quadric function $\phi(z,v) = (z_2 + v_2)^{1/2}$
- inverse multi-quadric $\phi(z,v) = \frac{1}{(z_2 + v_2)^{1/2}}$ but the most common in control applications is the Gaussian form $\phi(z,v) = \exp\left(-\frac{z^2}{v^2}\right)$

Similarly as MLP networks, RBF networks also represent a type of function expansion as can be seen from (2.13). In the case of Gaussian RBF networks they can be viewed as local function expansions. RBF networks also possess a universal approximation property, which has been proven, for example, in [Girosi and Poggio, 1990; Park and Sandberg, 1991]. That is, an RBF network can uniformly approximate to any desired accuracy any continuous function $f: R^n \rightarrow R^p$ over a compact set $K \subset R^n$, provided there are a sufficient number of nodes in the hidden layer. Again, this is an existence result not specifying how the parameters of the network can be obtained for a particular approximation task.

Centres of the basis functions need to be appropriately densely placed in the input space, which is due to the fact that, they have significant output only locally. Therefore, the RBF networks suffer from the "curse of dimensionality". For larger dimensions of the input space, the required number of nodes becomes prohibitively large.

2.4.4 Learning in RBF Networks

Output of the RBF network (2.13) is a linear function of the output weights w_{ji} but the centres ξ_i and width parameters v_i appears in the output equation nonlinearly. If the learning problem was formulated as a minimisation of some output error criterion w.r.t the full set of network parameters consisting of w_{ji}'s, ξ_i 's and v_i's, the situation would be very similar to learning in MLP networks, that is, due to the output of the network being a non-linear function of the estimated parameters, a non-linear minimisation task would involve risk of local minima and other problems.

However, advantages can be gained by exploiting the specific structure of RBF networks. A common strategy for the training of RBF networks is first fix to the centres of the hidden nodes. A variety of more or less heuristic methods has been developed.

One natural approach is to place nodes on a regular mesh in some region of the input space. For example, if the RBF network is used as a part of the controller, the mesh can cover entire admissible operating range of the plant. One such method is described the density of the mesh is chosen based on in [Sanner and Slotine, 1992]. In this paper, the density of the mesh is chosen based on the arguments of sampling theory and by making certain assumptions about the spatial Fourier transform of the unknown approximated function. The general problem with number of nodes required, which grows rapidly with placing centres on a mesh is the big tile dimension of the input space. Some other techniques place the centres around the measurement points (for example [Chen and Billings, 1992]. Such approaches in turn do not appear to be well suited for adaptive control, as it is difficult to tell ill advance where the operating point of the plant will move.

After the centres and widths have been fixed, the best values of the output weights need to be determined. Now, the estimated parameters appears linearly in the network output

$$y_j = \sum_{i=1}^{N} w_{ji}\phi_i(x) \qquad j=1,\ldots,p$$

where $\phi_i(x)$ are fixed functions of the input. Least Squares formulation of the estimation problems for linear-in-parameters models gives an objective function, which is a positive definite quadratic form in the unknown parameters. Therefore, there is a guarantee of finding a global minimum in a reliable manner. For example: the orthogonal least square algorithm for RBF networks is described in [Chen and Billings, 1992]. This property is probably the main advantage of RBF networks.

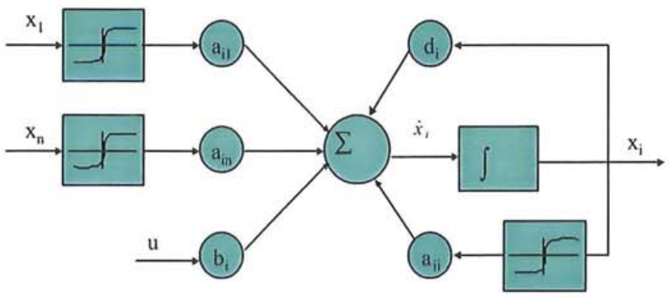


Fig. 2.5: Dynamic neurone of a Hopfield Neural Network

2.5 Recurrent Networks

Recurrent networks were originally introduced by [Hopfield, 1984] for the pattern recognition purposes serving as content addressable memories. The equilibrium points of the network are chosen such that they represent the uncorrupted patterns. A stable equilibrium point is surrounded by a region of attraction; that is, all initial conditions from a certain neighbourhood will converge to this equilibrium. Then, initial conditions belonging to this basin of attraction that corresponds to a pattern contaminated with noise will result in a steady state, which reflects the correct pattern.

2.5.1 Hopfield Networks

A single neurone of a continuous time Hopfield network is shown in Fig.2.5. Dynamics of such neurones are described by the following set of differential equations

$$\dot{x}_i = d^i x_i + \sum_{j=1}^{n} a_{ij}\sigma(x_j) + b_i u \qquad i=1,\ldots,n \tag{2.14}$$

where x_i are state variables of neurones, u is the input, d_i are negative constants describing linear part of the neurone dynamics and a_{ij} are the weights of the non-linear part. $\sigma(.)$ is a sigmoid function. The above set of equations can be described in a matrix-vector form as

$$\dot{x} = Dx + A\sigma(x) + Bu \tag{2.15}$$

where $x \in R^n$ is the state vector and $\boldsymbol{D} = diag(d_1,\ldots,d_n)$, $A \in R^{nxn}, B \in R^{nx1}$ are the matrices of weights. A discrete-time version of the Hopfield-type network is described by

$$x(k+1) = Dx(k) + A\sigma(x(k)) + Bu(k)$$

with state vector $x \in R^n$ and weight matrices $A \in R^{nxn}, B \in R^{nx1}$ Elements of the diagonal matrix **D** satisfy $-1 \le d_i \le 1$.

2.5.2 Approximation of Dynamical Systems by Recurrent Neural Networks

Most of the theoretical work on dynamic neural networks was motivated by their applications for content addressable memories and research was focused on stability of equilibrium points and convergence of network trajectories to the equilibrium. The ways of partitioning the state space into appropriate attraction regions are actively researched, see for example the paper [Sundharsanan and Sundareshan, 1991]. Only relatively recently, results showing abilities of dynamic Hopfield-type neural structures to approximate dynamic systems appeared in the literature [Funahashi and Nakamure, 1993; Sanchez, 1994; Jin *et al.,* 19951; Delgado *et al.,* 1995].

A natural way to define the approximation ability in the context of modelling of a dynamical system in the state-space representation is the following. Let the system to be approximated is described by:

$$\dot{q} = f(q,u) \tag{2.16}$$

where $q \in R^S$ is a state vector, $u \in R^m$ is the input vector. The approximating system is given by

$$\dot{x} = g(x,u) \tag{2.17}$$

where $x \in R^n$ is its state vector. Dimension n of the state vector x of the approximating system can in general be larger than s - state dimension of the modelled system. The approximation can be realised by choosing s states from the state vector of (2.17) to be the ones, which are supposed to model the behaviour of the original state vector **q**. Thus the s-dimensional dynamical system could be embedded into a higher n-dimensional one. Let us denote by x_s a vector composed of s states of (2.17). Then it is natural to require that for every initial condition ***q*** *(0)* (in a certain compact set) and input *u(t)* belonging to some class of signals (for example bounded), for an arbitrary $\in > 0$ the model (2.17) can be constructed such that $\max_t \|q(t) - x_S(t)\| < \in$ for an appropriate initial condition $x(0)$ of the model, over some time interval $0 \le t \le T$.

The following result in [Hirsch and Smale, 1974] provides a good starting point for constructing such dynamical models.

Lemma 1: Let $f, \hat{f} : D \to R^S$ be Lipschitz continuous mappings, where D is an open subset of $\mathbf{R}^s$. Let L be a Lipschitz constant of f. Suppose that for all $q \in D$ the following holds

$$\left\| f(q) - \hat{f}(q) \right\| < \in \tag{2.18}$$

If $q(t), \hat{q}(t) q(t)$ are solutions of

$$\dot{q} = f(q) \tag{2.19}$$

$$\dot{\hat{q}} = \hat{f}(\hat{q}) \tag{2.20}$$

respectively, on some time interval J, such that $q(t_0) = \hat{q}(t)$ then it holds that

$$\left\| q(t) - \hat{q}(t) \right\| \leq \frac{\in}{L} (\exp L\left| t - t_0 \right| - 1) \tag{2.21}$$

Therefore, over any finite time interval the trajectory approximation (2.21) can be made arbitrarily good by taking a suitably good approximation of f (i.e., $\in$ small enough in (2.18)). Thus, the dynamical model can be constructed based on the approximation of the static state function f and using integrators. It would be therefore natural to build dynamic models by utilising universal approximation capabilities of the static feed-forward networks with at least one hidden layer (as discussed in Section 2.3). Such a general model of a dynamical system, including also the output function $y = h(x)$, is shown in Fig.2.8.

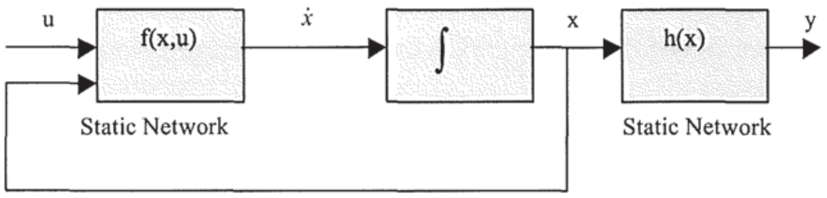


Fig.2.6: Model of a general dynamic system based on static network

However, for many purposes models of the form shown in Fig.2.6 might not be very manageable because mathematical analysis of their behaviour appears quite difficult. For example, one of the basic issues would be a formulation of stability criteria in terms of the weighting values. Due to complexity of a static network with a number of hidden layers such problem would be extremely difficult. Therefore, it is of significant interest for control and identification purposes to try to obtain structurally simpler dynamical models. Hopfield-type networks present an interesting case because their structure is relatively simple and there are a number of results concerning their stability.

The first result showing ability of Hopfield-type networks to approximate dynamical systems appears to be the one given in [Funahashi and Nakamura, 1993]. The paper considers Hopfield networks (2.14) without input and with the same linear dynamics in all neurones, that is, $d_i = d$, $i=l,\dots,n$

$$\dot{x} = dx + A\sigma(x) \tag{2.22}$$

The sigmoid function is chosen as $\sigma(x) = \dfrac{1}{1+e^{-x}}$

However, in the derivations, there is nothing preventing use of hyperbolic tangent function instead. Some neurones, whose states provide the states of the modelled system, are called the output units and rest is called hidden units. The following theorem is shown in that paper

Theorem 2: [Funahashi & Nakamura, 1993] *Let D be an open subset of R^s, R^S, $f : D \to R^S$ be a C^1 mapping, and $\widetilde{K}$ be a compact subset of D. Suppose that there is a subset $K \subset \widetilde{K}$ such that any solution **q(t)** is an initial value **q(O)** in **K** of an ordinary differential equation*

$$\dot{q} = f(q), \qquad\qquad q(0) \in K \tag{2.23}$$

*is defined on $J = [0, T](0 < T < \infty)$ and **q(t)** is included in $\widetilde{K}$ for any $t \in J$. Then, for an arbitrary $\in > 0$, there exists an integer **N** and a recurrent neural network (2.22) with s output units and **N** hidden units such that for a solution **q(t)** satisfying (2.23) and an appropriate initial of the network,* $\max_t \|q(t) - x_S(t)\| <\in$ holds, *where $x_s(t)=[x_1(t),\dots,x_s(t)]^T$ is the state of output units of the network.*

The proof of the above theorem is based on the universal approximation capabilities of a static network with a single hidden layer. Due to Theorem 1, there exists an integer ***N***, a *s x **N*** matrix W^1, a *N x s* matrix W^2 and an N-dimensional vector ***z*** such that on some appropriate compact set: $\max_q \left\| f(q) - W^1\sigma(W^2 q + z) \right\| <\in$.

Now, Lemma 1 could be readily applied. However, function $\hat{f}(q) = W^1\sigma(W^2 q + z)$ is not in the form matching the equation (2.22). The essential idea of the proof is to expand the state by defining an extra N-dimensional state vector $r = W^2 q + z$. In this way a recurrent network of the form (2.22) with $s+N$ neurons is obtained. Thus an s-dimensional dynamical system is embedded into a higher $s+N$ dimensional one. The network constructed in this way has connections between hidden units and from hidden units to output units but no connections from output units to hidden ones.

In the paper [Delgado *et al.*, 1995], a theorem is given stating that a dynamical system

$$\dot{q} = f(q,u) \tag{2.24}$$

$$y = h(q) \tag{2.25}$$

where $q \in R^s, u \in R, y \in R^P$, can be approximated by a recurrent network

$$\dot{x} = Dx + A\sigma(x) + Bu \tag{2.26}$$

where $x \in R^n, D \in R^{nxn}, A \in R^{nxn}$ and $B \in R^{nx1}$. The construction of the network exploits the same idea of extending the state as proposed in [Funahashi and Nakamura, 1993]. Thus, the state of the network consists of a part, which is supposed to model the original state $\boldsymbol{q}$ and an extra auxiliary part $x^T = [\hat{q}^T, r^T]$. The output of the network is constructed as $\hat{y} = h(\hat{q})$.

There is no rigorous proof given in [Delgado *et al.*, 1995], but rather a sketch of network construction. However, as the general idea is very close to that in [Funahashi and Nakamura, 1993] there appears to be no obstacle in proving such theorem. The network (2.26) has a more general structure than a Hopfield network (2.15) as its matrix $\boldsymbol{D}$ is full rather than diagonal as in (2.15).

A somewhat different way of using recurrent networks for approximation of dynamical system is presented in [Sanchez, 1994]. The network used in that paper has a diagonal structure, that is, there is no connections between neurones.

$$\dot{x} = Dx + A\sigma(x) + \sigma(u) \tag{2.27}$$

$$y = C\sigma(x) \tag{2.28}$$

where $\boldsymbol{D} = diag(d_1,\dots,d_n)$, $A = diag(a_1,\dots, a_n)$ and $C \in R^{1xn}$. The non-linear function is chosen as $\sigma(x) = \tanh(x)$.

The following condition is imposed: $-d_i > a_i$

The theorem states that: given a non-linear system represented as a time-invariant, causal and continuous operator *Y*, which has fading memory and for $|u|$ small enough, there exists a set of values (*n*, **D**, **A**, **C**) such that

$Yu(t) - \hat{F}u(t) < \in, \qquad \forall \in > 0, \qquad \forall u(t) \in U$

where $\hat{F}$ is the operator associated with this neural network.

Unfortunately, the paper does not specify precise conditions for $|u|$ being small enough which strongly limits applicability of this result.

A result concerning approximation capabilities of discrete-time recurrent networks is given in [Jin *et al.*, 1995] for the networks of the form

$$x(k+1) = dx(k) + A\sigma(x(k) + Bu(k)) \tag{2.29}$$

$$y(k) = Cx(k) \tag{2.30}$$

where $x \in R^n$ is the state vector, $u \in R^m$ is the input vector and $y \in R^p$ is the output vector. $A \in R^{nxn}, B \in R^{nxm}$ and $C \in R^{pxn}$ are weight matrices. Scalar d satisfies $-1 \leq d \leq 1$. The following theorem is stated in this paper.

Theorem 3: [Jin, Nikiforuk & Gupta, 1995] *Let* $D \subset R^s$ *and* $U \subset R^m$ *be open sets,* $K_D \subset D$ *and* $K_U \subset U$ *be compact sets, and* $f: DxU \to R^s$ *be a continuous vector-valued function which defines the following non-linear system*

$$q(k+1) = f(q(k), u(k)) \tag{2.31}$$

where $q \in R^s$ and $u \in R^m$, *with an initial state* $q(0) \in K_D$. *Then, for an arbitrary number* $\in > 0$ *and an integer* $0 < I < \infty$, *there exists and integer n and a recurrent network of the form (2.29), (2.29), (2.30) with an appropriate initial state x(0) such that for any bounded input* $u: R^+ \to Ku \max_{0 \leq k \leq I} \|q(k) - y(k)\| < \in$

First, a discrete-time equivalent of Lemma 1 is shown in the paper and, similarly as in [Funahashi and Nakamura, 1993], the proof of the above result takes the approximation capabilities of a static network with a single hidden layer as a starting point. Again, by extending the state dimension, an approximation of the s-dimensional dynamical system is obtained by embedding it into a higher dimensional one.

2.5.3 Useful Properties of Dynamic Neural Models

There appears to be a number of reasons why models based on dynamic neural networks can be useful for controller design in cases when there is a need for a dynamic model of the controlled non-linear system. Although Hopfield type networks of the form (2.15) or slightly more general (2.26) are non-linear dynamical systems, their nonlinearity is, loosely speaking, not too severe. With state equation defined by a combination of a linear function and a monotone non-linear one, such state space models can be viewed as a first step from linear to non-linear models. This relative simplicity allows development of mathematical analysis techniques for the networks themselves and consequently for the control systems based on such models.

For example, some quite mature results exist pertaining to the stability of networks described by (2.15), see [Perfetti, 1993; Forti *et al.,* 1994; Forti and Tesi, 1994; Fang and Kincaid, 1996b; Fang and Kincaid, 1996a; Jin *et al.,* 1994a; Jin and Gupta, 1996]. Due to boundedness of sigmoid activation functions, for both (2.15) and (2.26) it is sufficient for boundedness of solutions that the linear part of the function on the right-hand side of state equation (i.e. $\boldsymbol{D}_x$) corresponds to a stable linear system. Details of this useful properties can be found somewhere else.

Hyperbolic tangent function: $y = \tanh(x)$ can be expressed in the following way: $y = \omega(x)x$
where coefficient $\omega(x)$ (dependant on x) is defined as

$$\omega(x) = \begin{cases} \dfrac{\tanh(x)}{x} \rightarrow if, x \neq 0 \\ 1 \rightarrow if, x = 0 \end{cases}$$

Coefficient $\omega(x)$ has the property that it belongs to a bounded interval $\omega(x) \in (0,1)$. The whole non-linear part of the right-hand side of state equation in (2.15) or (2.26) can thus be represented by

$$A\,\sigma(x) = A\Omega x$$

where Ω is a diagonal matrix $\Omega = diag(\omega_1,..., \omega_n)$

In [Suykens *et al.,* 1995a; Suykens *et al.,* 1996], it is exploited in derivation of a Linear Fraction Transformation representation (used in robust control design) of general dynamic neural models. Using this approach, the neural state space models can be interpreted as nominal linear systems with bounded non-linear feedback perturbation.

An interesting result, related to the identification of dynamic neural models, is presented in [Albertini and Sontag, 1993a], where a few variations of the Hopfield type architecture, that input-output behaviour of the network uniquely determines its weights (up to re-labelling of neurones and sign reversals). This implies unique identifiability of network parameters with an appropriate choice of an identification experiment.

2.5.4 Learning in Dynamic Neural Networks

As the recurrent networks incorporate feedback, their learning is qualitatively different from learning in static networks. The neural network literature distinguishes between two general classes of learning problems in recurrent networks. The objective of the *fixed-point learning* is for the network to reach a prescribed set of equilibria. The steady state matching is required while the only condition on transients is that they die out, that is, the equilibria are stable. A more general case is that of *trajectory learning*, when the output of the network is required to follow some time trajectory. In both cases, whether, it is fixed point or trajectory learning, modification of weights employs gradient-based algorithms reducing some measure of error. Thus, the basic issue here is that of gradient calculation in dynamical systems. Many algorithms for recurrent networks training have been proposed under various neural networks literature, often with quite confusing names.

However, problem of gradient calculation in dynamical systems is not specific to the neural network field and solutions to it have been in existence for quite some time now. Early adaptive control techniques were based on updating controller parameters using gradient descent, see for example the introductory discussion in [Narendra and Annaswamy, 1989]. A very good unifying treatment of different learning algorithms for recurrent networks, presented in the general context of gradient calculation in dynamical systems, is given in [Baldi, 1995]. Consider a general dynamic system:

$$\begin{aligned} \dot{x} &= \hat{f}(x, w, u) \\ y &= h(x, w) \end{aligned} \tag{2.32}$$

where $x \in R^n$ is the state vector, $u \in R^m$ the input, $y \in R^p$ the output and **w** is the vector of parameters. Naturally, each of the continuous-time recurrent networks discussed in the preceding sections can be described in such a form.

2.5.5 Fixed Point Learning

Let us assume that the system (2.32) is stable and that stability is maintained during the learning process. For fixed initial conditions and fixed input, system (2.32) converges to an equilibrium point $\overline{x}$ which is a function of the parameter vector **w**. The output of the system in steady state is thus also a function of the parameters, $\overline{y} = h(\overline{x}(w), w)$. The objective is to modify the set of parameters in such a way that the steady state output $\overline{y}$ is as close as possible to the prescribed pointing y^*. This is done by minimising some error measure, which is usually chosen as a quadratic function

$$E = \frac{1}{2}(\overline{y} - y^*)^T (\overline{y} - y^*)$$

The equilibrium $\overline{x}$ point satisfies the steady-state relation

$$0 = \hat{f}(\overline{x}, w, u) \tag{2.33}$$

The derivative of the error function with respect to a single weight ω_i is given by

$$\frac{\partial E}{\partial \omega_i} = \sum_j \frac{\partial E}{\partial \overline{y}_j} \frac{\partial \overline{y}}{\partial \omega_i} = \left[\frac{\partial E}{\partial \overline{y}}\right]^T \frac{\partial \overline{y}}{\partial \omega_i}$$

The derivative of the steady state output is obtained as

$$\frac{\partial \overline{y}}{\partial \omega_i} = \frac{\partial h(\overline{x}, w)}{\partial \omega_j} + \frac{\partial h(\overline{x}, w)}{\partial x} \frac{\partial \overline{x}}{\partial \omega_i}$$

The first element in the above sum results from explicit dependence of the output function h on ω_i. The second element results from influence of ω_i on the dynamics of x. In case of recurrent neural networks one of them is normally equal zero (a weight is either in the output equation or in the dynamical part). Calculation of the expression $\frac{\partial \overline{x}}{\partial x}$ is done by differentiation of the steady-state relation (2.33). Denoting by $\hat{f}_j$ a single element of the vector-valued function $\hat{f}$, we obtain

$$0 = \frac{\partial \hat{f}_j}{\partial x} \frac{\partial \overline{x}}{\partial \omega_i} + \frac{\partial \hat{f}_j}{\partial \omega_i}$$

which in a matrix-vector notation is

$$0 = \frac{\partial \hat{f}}{\partial x} \frac{\partial \overline{x}}{\partial \omega_i} + \frac{\partial \hat{f}}{\partial \omega_i}$$

where $\frac{\partial \hat{f}}{\partial x}$ denotes a Jacobian of function $\hat{f}$ w.r.t x. Thus, from the above we obtain

$$\frac{\partial \overline{x}}{\partial \omega_i} = -\left[\frac{\partial \hat{f}}{\partial x}\right]^{-1} \frac{\partial \hat{f}}{\partial \omega_i}$$

Finally, the gradient of the error function is given by

$$\frac{\partial E}{\partial \omega_i} = \left[\frac{\partial E}{\partial \overline{y}}\right]^T \left(\frac{\partial h}{\partial \omega_i} - \frac{\partial h}{\partial x}\right)\left[\frac{\partial \hat{f}}{\partial x}\right]^{-1} \frac{\partial \hat{f}}{\partial \omega_i}$$

Expressions $\frac{\partial \hat{f}}{\partial x}, \frac{\partial \hat{f}}{\partial \omega_i}, \frac{\partial h}{\partial \omega_i}$ and $\frac{\partial h}{\partial x}$ are calculated at the equilibrium point $\overline{x}$. The weights are usually updated in the direction of steepest descent according to

$$\frac{d\omega_i}{dt} = -\lambda \frac{\partial E}{\partial \omega_i}$$

Derivation of a fixed point learning scheme for a discrete-time recurrent network follows the same principles.

2.5.6 Trajectory Learning

In the trajectory learning, the goal is to obtain such values of weights that the network output *y(t)* follows a desired trajectory *y*(t)*, over some interval $[t_0, t_I]$. This approach is therefore directly applicable to the problem of obtaining a model of a dynamic system based on a recurrent network. Rather than following a signal trajectory, the goal is then to find such values of weights that the network produces the same output trajectories as the modelled dynamical system when both are excited by the same input. The error function is defined as an integral of some measure of the instantaneous error, typically a quadratic function.

$$E = \int_{t_0}^{t_1} e(y(t), y^*(t))dt = \frac{1}{2}\int_{t_0}^{t_1} y(t) - y^*(t))^T (y(t) - y^*(t))dt$$

The gradient of the error function is calculated as

$$\frac{\partial E}{\partial \omega_i} = \int_{t_0}^{t_1} \left[\frac{\partial e}{\partial \overline{y}}\right]^T \frac{\partial \overline{y}}{\partial \omega_i} dt \qquad (2.34)$$

Similarly as before,

$$\frac{\partial \overline{y}}{\partial \omega_i} = \frac{\partial h(x,w)}{\partial \omega_j} + \frac{\partial h(x,w)}{\partial x}\frac{\partial \overline{x}}{\partial \omega_i}$$

but now the expression $\frac{\partial x}{\partial \omega_i}$ reflects influence of changes in w_i on **x** over a certain time span, not only in the equilibrium point. It is computed from differential equation of the system (2.32), by applying partial derivative

$\frac{\partial}{\partial \omega_l}$ to both sides of (2.32) (assuming that $\frac{\partial}{\partial \omega_i}$ and $\frac{d}{dt}$ commute)

$$\frac{d}{dt}\frac{\partial x}{\partial \omega_l} = \frac{\partial \hat{f}}{\partial x}\frac{\partial x}{\partial \omega_l} + \frac{\partial \hat{f}}{\partial \omega_l} \qquad (2.35)$$

Both $\frac{\partial \hat{f}}{\partial \omega_l}$ and the Jacobian matrix $\frac{\partial \hat{f}}{\partial x}$ are evaluated at the current value of state *x(t)* and thus are time dependant. Therefore, the above system can be classified as Linear Time-Varying. If the initial condition $x(t_o)$ in (2.32) does not depend on the parameters w_i, as is usually the case, the initial condition for (2.35) $\frac{\partial \hat{f}}{\partial \omega_l}(t_0) = 0$. It has to be stressed that the derivation of (2.35) relies on the assumption that parameters w_l is constant.

The question is now how to obtain the solution of the differential equation (2.35). It has been shown in [Baldi, 1995] that amongst all learning algorithms proposed in neural literature so far, there are in fact only two distinct ways of computing this gradient.

One approach is to integrate equation (2.35) forward in time. That means that for every single parameter w_i a set of n differential equations needs to be numerically integrated. This technique is called the *sensitivity* method. Its use in the context of neural networks has been well described in [Narendra and Parthasarathy, 1991]. The basic advantage is that this method is readily applicable for on-line systems. The main disadvantage of this approach is the heavy computational burden.

The other approach is to use the so-called *adjoint* method. Instead of integrating forward in time one set of equations (2.35) for each weight w_i, solutions of (2.35) can be obtained by using a solution of an auxiliary n-dimensional system, the adjoint system, integrated backwards in time. This method exploits the fact that for each of the weights w_i, the autonomous part of the corresponding sensitivity equation (2.35) the same, since it is based on the Jacobian of the original system $\frac{\partial \hat{x}}{\partial x}$. This method is well described in [Baldi, 1995]. It is also shown there that an on-line version of this method can be obtained but it requires inverting an $n \times n$ matrix.

Updates of the network parameters can be performed either continuously or after the whole integration time $[t_0, t_l]$. This point is further elaborated in Section 4.3, where an adaptive control technique using recurrent neural networks is presented.

CHAPTER 3: NEURAL NETWORKS CONTROL STRATEGIES

3.1 Introduction

The objectives of the chapter are:

- to present a comprehensive investigation of the various adaptive control schemes using neural networks trained by supervised learning.
- to classify the control schemes in an organised structure.
- to present discussions and critical reviews of these neural network controllers.
- to propose possible control structures.
- to present an overview of control applications using neural network.

At present there are a number of survey papers and books that exist on using neural network for control. For examples, in [Hunt *et al.*, 1992; Miller *et al.*, 1990; Werbos, 1990b; Warwick *et al.*, 1992; Irwin *et al.*, 1995; Agarwal, 1994; Sontag, 1993; Brown and Haris, 1994; Hunt *et al.*, 1995; Pham and Liu, 1995; Balakrishnan and Weil, 1996; Omidvar and Elliott, 1997; Lewis *et al.*, 1999] and [Haykin, 1994 and 99]. Also a number of mini-surveys with suggested methods or possible applications can be found in [Tolle, 1994; Narendra Parthasarathy, 1990; Slotine and Sanner, 1993] and [Special Editions of the neural network control system Magazine (1990)]. However, most of these surveys are not comprehensive enough to cover the wide and fast developing scope of this field. For example in [Werbos, 1990b] five methodologies are suggested and [Hunt *et al.*, 1992] add four more methodologies to the list of Werbos.

Others use twofold classifications, such as direct and indirect controls. Two-fold classifications are standard methods for classifying conventional adaptive control schemes. However, the innovations of control engineers have developed beyond the scope of a single unified twofold classification. Nevertheless, some of these twofold classifications still useful terms to specify control structures. Hence they are discussed here in detail.

The aim of this discussion is to give an overview of all possible control schemes, as well as allowing analysis and new control schemes that are easily categorised into multi-levels. The first level divides all control schemes into two broad groups namely; control schemes controlled by the neural networks only and control schemes where neural networks work with other controllers. The latter group, which are called hybrid strategies, refer to those control schemes where neural networks are used as an aid to enhance the performance of conventional control strategies or vice versa. Some conventional control strategies are referred to as classical control, adaptive control, optimal control, expert control and fuzzy control. The main motivation for hybrid strategies is that an effective combination of neural network and other controllers might improve the control performance. The former group, which called non-hybrid strategies, refers to control schemes where the model and / or the controller are implemented using neural networks only.

Since neural networks need to be trained by signals, it is appropriate to define the second level of classification intervals of how the networks are being trained. For example in supervised learning, how is the error generated to train the neural networks? Then the second level is categorised as a weight updating process based upon:

(a) Control signals

(b) Desired output signals

(c) Feedback controller output signals

These second level categories can be extended to other methods of training neural networks. At the moment these categories, have captured most of the existing neural network controllers trained by supervised learning.

To facilitate simple analysis and understanding of the various control schemes in the literature, terms are used for the classification levels that are already familiar to researchers working in this field. Fig.3.1 shows multi-level categorisation of neural networks control strategies. Notice that most of these control structures and their terminologies are borrowed from the field of conventional adaptive control.

This chapter can be divided into four parts. The first part presents all possible twofold classifications of neural networks for control schemes. The second part discusses the non-hybrid neural networks control strategies. These are covered in sections 3.3 and 3.4. The third part discusses all hybrid neural networks control strategies. These are in sections 3.5, 3.6 and 3.7. The fourth part looks into some of the applications in control using neural networks.

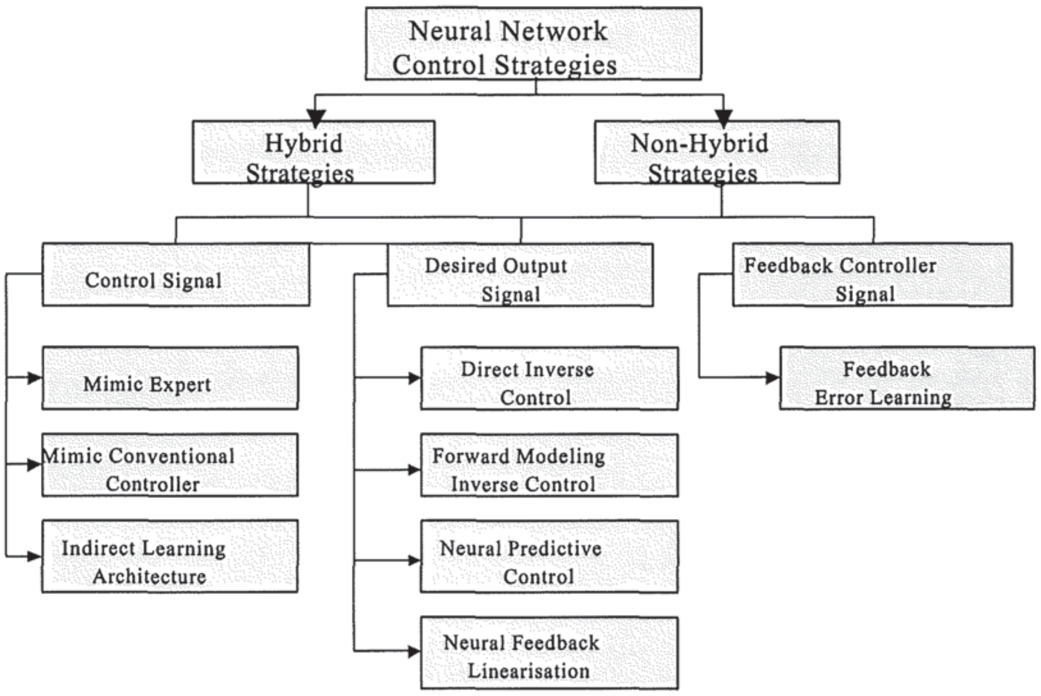

Neural Network Control Strategies
Hybrid Strategies
Non-Hybrid Strategies
Control Signal
Desired Output Signal
Feedback Controller Signal
Mimic Expert
Mimic Conventional Controller
Indirect Learning Architecture
Direct Inverse Control
Forward Modeling Inverse Control
Neural Predictive Control
Neural Feedback Linearisation
Feedback Error Learning

Fig.3.1: Multi-level categorisation of neural network control strategies

3.2 Two-fold Classifications

In the literature there are a number of twofold classifications of neural networks control strategies. Some suggested possible twofold classifications and those described in the literature are listed below.

(a) **Goal and not goal oriented** [Hunt *et al.*, 1992]. The original concept of goal and not goal-oriented classifications depends on how the neural network is trained. If training is based on the desired signal, it is known as goal oriented, otherwise it is not goal oriented. It is difficult to draw a strict line on some neural network controller training approaches as being either goal oriented or not goal oriented. For example, neural network controllers that mimic human experts or neural networks trained by using the signal output from the feedback controller, such as feedback error learning, are not very obviously goal or not goal oriented.

(b) **Closed loop and open loop**. Closed loop and open loop are common terminology used in conventional control systems. Most of the neural network control schemes discussed can be designed as either closed loop or open loop. For closed loop form we refer to the case where the neural network controller inputs consist of the past errors signal$e_s(k), e_s(k-1),..., e_s(k-n+1)$. The open loop does not have these feedback error signals; instead the desired plant output and the past states of the plant are the input to the neural network. An example of a neural network controller in closed-loop form can be found in [Bleuler *et al.*, 1990].

(c) **Feed-forward and feedback control.** Like closed loop and open loop, feed-forward and feedback control are common terminologies used in conventional control systems. For a neural network feedback controller, the control structure is connected in a closed-loop form (see Fig.3.6). The inputs to the networks are the error from the difference between the desired signal and the actual plant output signal and sometimes include the past plant outputs [Porter and Liu, 1994]. The inputs to the feed-forward controller are the desired signal and the past plant output and input values (see Fig.3.5). Technically feed-forward and feedback control are similar to closed loop or open loop control structure.

(d) **Reference model and without reference model control**. For reference model control the desired performance of the system is specified through a stable reference model [Hunt *et al.*, 1992]. Reference model control has been widely used in linear adaptive control application [Astrom and Wittenmark, 1989]. The control system attempts to make the plant output $y_p(k)$ match the reference model output asymptotically, that is,

$$\lim_{k \to \infty} \left\| y_p(k) - y_m(k) \right\| \leq \epsilon \qquad (3.1)$$

for some specified constant $\epsilon \geq 0$. However, neural networks control schemes designed for regulation or servo (trajectory tracking) do not preclude the use of a reference control model.

(e) **Direct and indirect control schemes**. These two schemes are formulated in the conventional adaptive control [Astrom and Wittenmark, 1989]. In direct adaptive control, the parameters of the controller are directly adjusted to reduce the output error between the plant and the desired output. In indirect adaptive control, the parameters of the plant are estimated and the controller is chosen assuming that the estimated parameters represent the true values of the plant parameters. A direct adaptive control scheme builds an explicit model of the desired controller, whereas an indirect scheme produces a model of the plant and synthesises the control law, using a predefined optimisation or inversion calculation. This twofold classification is used in neural networks control structures such as [Psaltis *et al.*, 1988]'s specialised learning (SL) structure can be classified as direct control schemes, whereas [Jordan and Jacobs 1990] put forward and inverse modelling that can be classified as an indirect control schemes.

(f) **Hybrid and direct types**. This is the method of classification used by [Kojima *et al.* 1994]. The hybrid type refers to a control scheme where the neural network is used to tune the parameters of the conventional controller. The direct type refers to the neural network as a controller constructed by learning the inverse dynamics of the control target.

(g) **Generalised and specialised learning**. Off-line training of the neural network model to emulate the plant is sometimes known as generalised learning (GL). This is because the neural network model is trained to emulate the plant over a generalised operating range where the controlled variable is expected to perform. For cases where the neural network model is only trained for a specific path or trajectory where the controlled variable will perform, it is known as specialised learning (SL).

(h) **Inverse and non-inverse control**. It is clear that most of the neural networks for control structures involve implicitly or explicitly training an inverse model of the plant. Fig.3.2 shows an example where the neural network controller is connected to perform inverse control. Here, the neural feed-forward controller has to perform inverse dynamic modelling of the non-linear plant. Most of the control schemes discussed in the following sections involve some form of inverse dynamic modelling of the plant. The disadvantages of inverse control are:

- the plant must be invertible
- non-minimum phase plant is inverse unstable [Slotine and Sanner, 1993; Narendra and Parthasarathy, 1990].

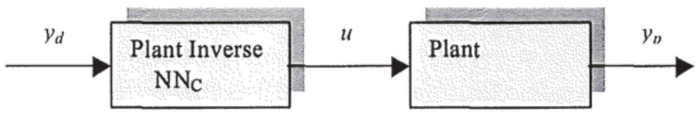


Fig. 3.2: Neural network controller in feed-forward connection

It is clear that the above two fold classifications are not comprehensive enough to cover the increasing diversity of different neural networks control strategies used. Furthermore some of the twofold classifications are closely related to one another. However, they may be useful to serve as sub-classifications or terms used for better understanding of a specific structure.

3.3 NON-HYBRID STRATEGY-CONTROL SIGNAL

3.3.1 Mimicking the human expert

For control problems, which are poorly defined or involve processes that are difficult to describe analytically, a human expert often arrives at a reasonable solution based on experience and intuition. The expert's work can partly or completely be relieved by storing in a neural network his/her valuable diagnosis and decision-making know-how. Two different approaches are possible in this domain.

1. The expert looks at the available information and decides an appropriate control action. The neural network then simply extracts a functional mapping between the used information and the expert's control actions.
2. The human expert first translates his/her 'feel' for the problem solving into concrete logical rules, which can then be embodied as a neural network using direct geometric methods.

Mimicking the human expert is essentially similar to the supervised control classified by [Werbos, 1990b]. It can be seen as a kind of neural expert system, which pays attention to what experts actually do instead of what they say they do. Werbos also pointed out that if a human can only perform the task at slow speed, in simulation, a mimicked neural network controller can learn to imitate the human and then operate more quickly. Further discussion of mimicking the human expert can be found in [Osuka *et al.*, 1989; Rezeka, 1995].

3.3.2 Mimicking a conventional controller

In this scheme, neural networks mimic a conventional controller [Damle *et al.*, 1994]. The neural network controller identifies the implicit function underlying the available controller. After learning, the neural network is applied to replace or support the conventional controller. Willems, 1993 called this scheme an imitation control (IC). **Why should one apply a neural network controller when a controller already exists?** Neural network controller should be applied when:

- The conventional controller is more computationally intensive than the neural network controller is.
- The conventional controller requires system state information that is difficult to achieve while neural networks may learn to identify a control rule easily.

To train a network to identity a controller, the network can learn with the system information that is used by the controller as input and the control signal from the controller output as the desired output. A mimic of a conventional structure is illustrated in Fig.3.3.

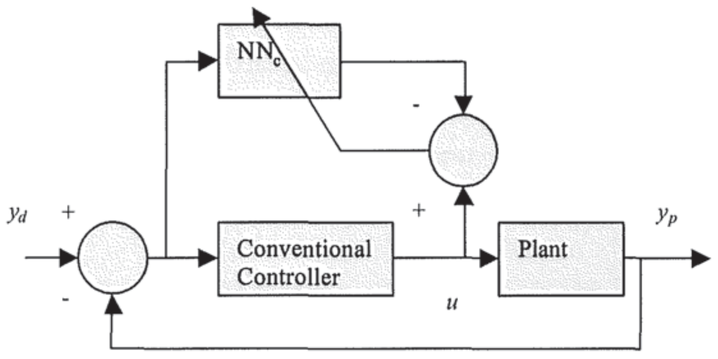


Fig. 3.3: Mimicked conventional controller

Willems (1993) reported that this type of neural control architecture could be:

1. Performed on-line, therefore the learning of the neural network does not interfere with the industrial system's process.
2. Learned without the input information, that is, the error, *e* that is available to the existing controller. The System State alone together with the response of the controller is sufficient to train the neural network.

The neural network controller after being trained can also be used as a decision support system. Neural networks applied in this way are sometimes referred to as implicit expert systems since the knowledge used by the controller is represented in an implicit sub-symbolic way. The disadvantage is that it delivers only an approximate control action.

3.3.3 Indirect learning architecture

Fig. 3.4 shows the indirect learning architecture [Psaltis *et al.*, 1988]. The input to the neural network is the plant output and the error is generated by comparing the neural network output and the control signal.

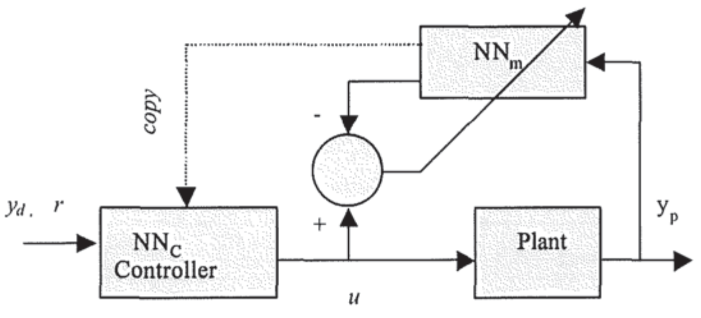


Fig. 3.4: Indirect learning architecture

The neural network can be trained on-line or off-line. For an off-line approach a training set can be obtained by generating inputs *u(k)* at random and observing the corresponding outputs $y_p(k)$. The inverse system is then trained by attempting to fit the reversed pair $(y_p(k),u(k))$. This is also known as general learning architecture [Psaltis *et al.,* 1988]. Once an inverse model is trained, the neural network is used as a controller in the feed-forward connection. For the on-line approach, after each iteration of training the neural network controller is merely a copy of the updated neural network model. Andersen *et al.* (1995)'s single net indirect learning architecture is the same as the on-line approach discussed here.

The disadvantages of indirect learning architecture are:

- Mimimizing the difference between actual plant input and the estimated plant input generated from the neural network inverse model does not necessarily minimise the error between the desired plant output and the actual plant output. Hence, minimisation of the output error $e_s(k)$ is not guaranteed. This problem is also known as not goal oriented training.
- Robustness problem. Sontag (1993) highlighted that such an approach would suffer serious robustness problems even for linear systems. For non-linear systems, there are additional difficulties although best inverse exist locally, so training on incomplete data cannot be expected to result in a good interpolation or 'generalization' capability, which after all is the main objective of learning control.
- Cannot be used for plant where a functional relationship does not exist in the input and output data, i.e., the relationship between *u(k-1)* and the signals input to the neural network is not one to many [Slotine and Sanner, 1993; Jordan and Rumalhart, 1992].

The possible advantages of using this control scheme are ease of implementation and only one neural network is needed.

3.4 NON-HYBRID STRATEGY - DESIRED OUTPUT SIGNAL

3.4.1 Direct inverse control

Direct inverse control was not popular in the late 80s and early 90s. This is because the unknown non-linear plant lies between the neural network controller and the system output error e_s [Narendra and Parthasarathy, 1990]. Figures 3.5, 3.6 and 3.7 show the possible structures of the direct inverse control scheme. The structure in Fig.3.7 is sometimes known as feed-forward control or specialised learning. The former name is used because the controller is placed before the plant with the reference signal input to the neural network controller; the latter name is used because the neural network controller is learning to control in a specific desired range. This structure is closely related to model reference adaptive control (MRAC) [Sontag, 1993]. For the closed loop direct inverse control (Fig. 3.6), the neural network controller is acting like a feedback controller.

Direct inverse control treats the plant as if part of the neural network output layer. Hence it faces the problem of an unknown plant Jacobian. Several methods are proposed to overcome this problem. For example, the use of plant Jacobian's sign [Saerens and Soquet, 1991; Ng and Cook, 1996c; [Zhang et. al., 1995], calculating the plant Jacobian from the model [Ng and Cook, 1995], using the Inverse Transfer Matrix Scheme [Chen and Pao, 1989] and the Alopex algorithm [Venugopal *et al.*, 1994].

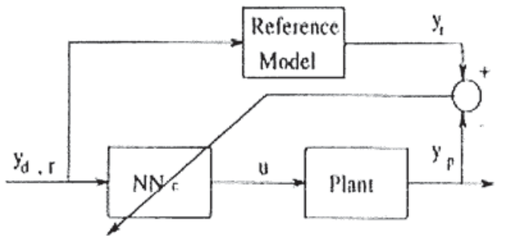


Fig.3.5: Direct inverse control with reference model

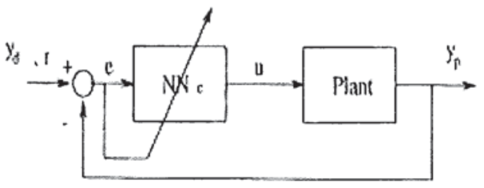


Fig.3.6: Direct inverse control (closed loop)

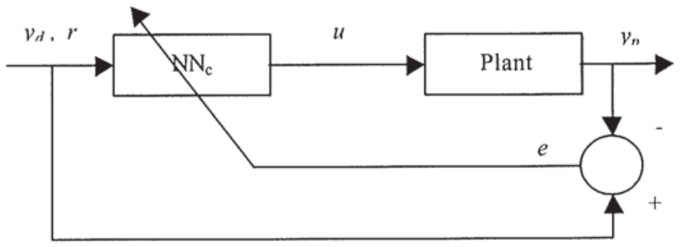


Fig.3.7: Direct inverse control (open loop)

Inverse transfer matrix scheme

This is suggested by [Chen and Pao, 1989)]. In this scheme, the error function to be minimized is chosen as:

$$E_c(k) = \frac{1}{2}[K_c e_s(k)]^2 \tag{3.2}$$

where K_c represents the inverse transfer relationship of the dynamics. The neural network updating is done by estimating the derivative of the error function with respect to the weights, that is,

$$\frac{\partial E_c(k)}{\partial W(k-1)} = -K_c^2 e_s(k)\frac{\partial y_p(k)}{\partial W(k-1)} = -K_c^2 e_s(k)J(k)\frac{\partial u(k-1)}{\partial W(k-1)} \tag{3.3}$$

When $K_c \approx [J]^{-1}$ the inverse transfer matrix scheme is obtained. Then

$$\frac{\partial E_c(k)}{\partial W(k-1)} = -K_c e_s(k) \frac{\partial u(k-1)}{\partial W(k-1)} \tag{3.4}$$

Equation (3.4) shows that Chen *et al.*'s approach attempts to minimise the neural network output error. In this scheme, the inverse differential gain K_c is better computed off-line. On-line computation of K_c may have numerical problems such as large change in the network weights.

3.4.2 Forward modelling and inverse control

In this approach, a neural network model is trained to emulate the plant. The neural network model is trained using the error $e_m(k) = y_p(k) - y_m(k)$. A training set can be obtained by generating inputs $u(k)$ at random and observing the corresponding outputs $y_p(k)$ and $y_m(k)$. Once trained, the neural network controller could be updated by using the information in the neural network model and the system error, commonly $e_s(k) = y_d(k) - y_p(k)$, in the following three methods:

Method 1: Back-propagation through the model

This method uses the pre-trained neural network model to back-propagate the system error back to the neural network controller. The neural network controller then learns the desired control law by the BP algorithm. Jordan and Jacob called this forward and inverse modelling. As explained in [Jordan Jacobs (1990)], The plant Jacobian matrix $J(k) = \frac{\partial y_p(k)}{\partial u(k-1)}$ can not be assumed to be available a priori, but can be estimated by back-propagation through the forward model. The system error $e_s(k) = y_d(k) - y_p(k)$ is back propagated through the pre-trained model. Hence the method requires an accurate pre-trained neural network model. Furthermore, it is assumed that plant parameters do not change when the neural network model is trained and fixed.

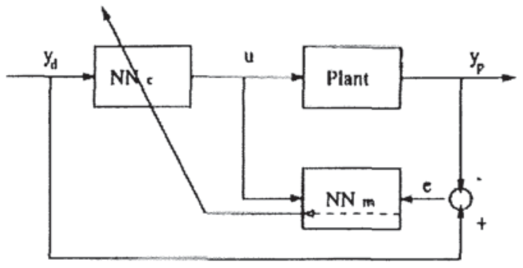


Fig.3.8: Forward and inverse modelling

The on-line adaptive neural network based controller (OANNC) proposed by [Yang and Link (1994)] is quite similar to forward and inverse modelling. One of the differences in [Yang and Linkens, 1994] method is that the system error for training the neural network controller is obtained from desired output minus the neural network model instead of the actual plant output, hence the system error is $e_s(k) = y_d(k) - y_m(k)$. Yang and Linkens (1994)'s approach of minimising system error may not mean reducing the plant output error. This could be problematic when the plant parameters change after a period of time.

Method 2: Back-propagation through time (BPTT)

The key differences between BPTT and Jordan's forward and inverse modelling are:

1. The system error at time step k is back propagated through each of the k time steps to update the neural network controller weights.
2. The neural network controller is trained off-line.
3. The neural network controller weights are updated after k time steps.

The basic concept of BPTT is as follows. The first task is to develop a model of the non-linear plant. The training of the plant model or neural network emulator consists of applying a sequence of random input $u(k)$ to the plant and observing the plant output. The neural network emulator is trained using the BP algorithm. After the neural network emulator is trained, the following procedure is used to train the neural network controller:

(a) The neural network controller receives $x(0)$ and generates $u(0)$. The control $u(0)$ generates the new plant states $x(1)$. Without updating the weights in both neural networks, the controller receives $x(1)$ and generates $u(l)$ and so on for a maximum number of steps specified by the designer, say $x(k_N)$, where k_N is the specified number of time steps in the trials; or until the plant output states reach the desired states.

(b) The state $x(k_N)$ is compared with the desired final state $x_d(k_N)$. The difference is back propagated through the neural network emulator to update the weights of the neural network controller. However, instead of being updated just once, the neural network controller is updated k_N times, as if a large network with k_N copies of the neural network emulator and neural network controller were used. The weights of the neural network emulator are kept fixed. Fig.3.9 shows the neural network controller trained by back-propagation through time.

(c) The plant is then initialized at other states and steps 1 and 2 are repeated.

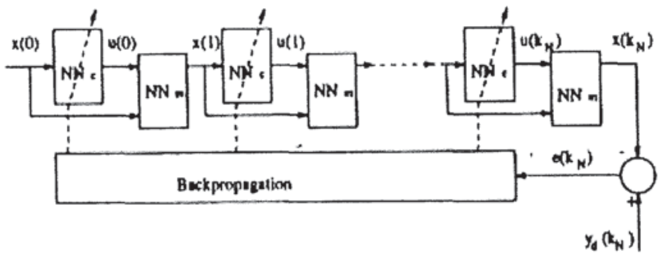


Fig. 3.9: Neural network controller trained by back-propagation through time

When acceptable tracking has been obtained using the neural network controller with the neural network model, the neural network controller may be brought on-line with the actual process. For BPTT to work well, the neural network emulator has to be a highly accurate representation of the plant. This has reduced the BPTT applicability. Fu (1994)'s Temporal Model and [Nguyen and Widrow 1990's] self-learning control systems are similar to BPTT approach (see Chapter 4).

Method 3: Narendra and Parthasarathy's approach

Narendra and Parthasarathy (1990) proposed a control structure, which is suitable for plant depending non-linearly on the control signal. In this approach, three neural networks are required. The identification stage consists of two neural networks. The third neural network is the controller (NN_c). The objective is to determine a control input so that $y_p(k)$ follows the desired output $y_d(k)$ or $y_r(k)$ asymptotically.

This approach can be summarised as follows:

1. Assume that a first order non-linear system is described by the equation $y_p(k+1) = f[y_p(k)] + g[u(k)]$ where f and g are unknown continuous functions and g has an inverse denoted by g^{-1}.
2. The output of the reference model is $y_r(k+1) = \beta_r[y_r(k) + r(k)$ and $\|\beta_r\| < 1$.
3. Estimate the function f and g using neural networks NN_f and NN_g respectively.

$$\hat{y}_p(k+1) = NN_f[y_p(k)] + NN_g[u(k)] \tag{3.5}$$

This estimation is carried out off-line using random inputs to the plant.

4. Estimate the inverse of g using NN_g to train the NN_c (see Fig.3.10).
5. Once NN_f, NN_g and NN_c are known $u(k)$ can be computed as:

$$u(k) = NN_c(NN_f(y_p(k)] + \beta_r y_r(k) + r(k)) \tag{3.6}$$

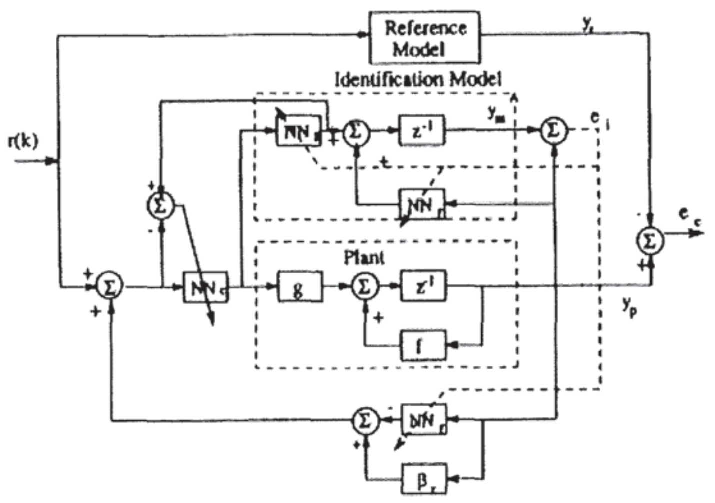


Fig. 3.10: Narendra and Parthasarathy's (1990) approach

The disadvantages to this approach are:

- More than one neural network is required (or just "three neural networks are required").
- Training of the neural networks has to be done off-line.
- The identification model can only approximate the inverse or forward model of the plant over a specified finite region.

3.4.3 Neural internal model control

Hunt and Sbarbaro (1991) suggested that neural networks could be incorporated into the conventional internal model control (IMC) structure proposed by [Economou and Morari, 1986; Morari and Zafirious, 1989].

In IMC, a plant model is placed parallel to the real plant. The difference between the plant and the model outputs is used for feedback purposes. If the model is a perfect representation of the plant, the feedback signal will be the plant disturbances and noise.

In neural internal model control (IMC), a neural network is trained to model the plant. Assuming that the neural network model is an ideal representation of the real plant, hence $y_m = y_p(k)$ then the closed loop system of Fig.3.11 is input-output stable if the neural network controller and plant are input-output stable. This assumes that the neural network controller has the right inverse of the neural network model. Then perfect control is achieved, $y_p(k) = r(k)$. IMC is configured in closed loop form. Other papers that look into using IMC are [Abdulaziz and Farsi, 1993; Sbarbaro *et al.,* 1993].

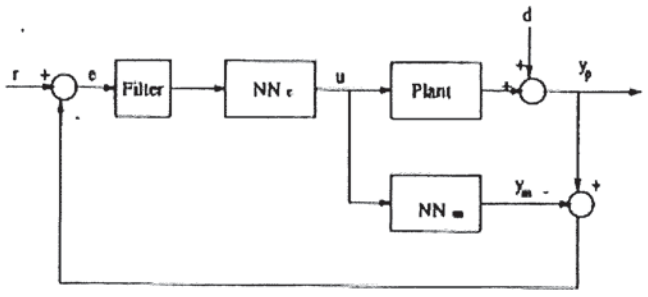


Fig. 3.11: Neural internal model control

3.4.4 Neural feedback linearisation

This approach stems from the conventional linearizing feedback control [Isidori, 1989] where the unknown plant is feedback-linearisable. The idea is to transform a state space model of the plant into new co-ordinates where non-linearities can be cancelled (fully or partially) by feedback. The major challenge in performing such cancellation is the need to know precise models of the non-linearities. In this case, the neural networks are used to model the non-linearities.

To illustrate this, consider an adaptive regulation problem for a SISO relative degree one plant [Chen and Khalil, 1992]

$$\begin{aligned} y_p(k+1) &= f[y_p(k),..., y_p(k-n_y+1), u(k-1),..., u(k-n_u)] \\ &+ g[y_p(k),..., y_p(k-n_y+1), u(k-1),..., u(k-n_u)]u(k) \end{aligned} \tag{3.7}$$

Choosing the state variables as

$$\boxed{\begin{array}{l} z_{11}(k) = y_p(k-n_y+1) \\ \vdots \\ z_{1n_y}(k+1) = y_p \\ z_{1n_y}(k) = y_p(k) \\ z_{21}(k) = u(k-n_u) \\ \vdots \\ z_{2n_u}(k) = u(k-1) \end{array}} \quad \text{one obtains the state space model} \quad \boxed{\begin{array}{l} z_{11}(k+1) = z_{12}(k) \\ \vdots \\ z_{1n_y}(k+1) = f[z(k)] + g[z(k)]u(k) \\ z_{21}(k+1) = z_{22} \\ \vdots \\ z_{2n_u}(k+1) = u(k) \\ y_p(k) = z_1 n_y(k) \end{array}}$$

Then with the feedback control:

$$u(k) = \frac{1}{g[z(k)]}[-f(z(k)) + y_d(k)] \tag{3.9}$$

the plant takes the following form:

$$
\begin{aligned}
z_1(k+1) &= Az_1(k) + By_d(k) \\
z_2(k+1) &= F(z_1(k), z_2(k), y_d(k)] \\
y_p(k) &= C_{z_1}(k)
\end{aligned}
\tag{3.10}
$$

where

$$
A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}; \; B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}; \; C = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{3.11}
$$

are controllable-observable triple. This is provided that the following assumption holds:

1. *f(z)* is smooth and vanishes at the origin.
2. *g(z)* is smooth and bounded from zero over *S*, a compact subset of $\Re^{n_y+n_u}$.
3. The plant is minimum phase.

Literature that include neural feedback linearisation are [Braak *et al.*, 1994; Peel *et al.*, 1994; Hancock and Fallside, 1993; Yesildirek and Lewis, 1994; Liu and Chen, 1993; Mccullough, 1992; Delgado *et al.*, 1995].

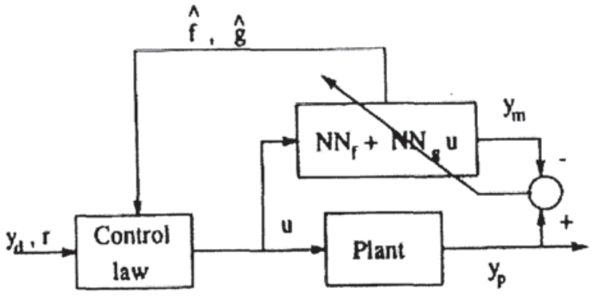


Fig.3.12: Neural Feedback linearisation

3.4.5 Neural predictive control

Neural predictive control (NPC) assumes that a neural network model is well trained to emulate the plant. The neural network model is then fixed. The objective is to calculate the control such that the error over the future horizon is minimised; that is, we consider the cost function of the following form:

$$
J = \sum_{j=N_1}^{N_2} [y_r(k+j) - y_m(k+j)]^2 + \sum_{j=1}^{N_u} \rho(k)[u(k+j-1) - u(k+j-2)]^2 \tag{3.12}
$$

where y_r, represents the output of the reference model (i.e., desired output) and y_m is the output of the neural network model. N_1 is the minimum output prediction horizon. N_2 is the maximum output prediction horizon and N_u is the control horizon. $\rho(k)$ is the move suppression factor or control weighting sequence. It is used to penalise excessive control effort. The first term of the cost function is a measure of the errors the model predicted and the desired future trajectory. The second term penalises excessive movement of the control variable. References to NPC can be found in [Saint-Donat *et al.*, 1991; Hunt *et al.*, 1992; Hunt and Sbarbaro ,1992; Joseph and Hanratty, 1993; Warwick *et al.*, 1995; Saint-Donat *et al.*, 1994; Willis *et al.*, 1992; Park and Cho, 1995; Draeger *et al.*, 1995].

The general NPC steps are as follows:

1. Predict the system output over the range of future times.
2. Assume that the future desired outputs are known.
3. Choose a set of future controls u, which minimize the future errors between the predicted future output and the future desired output.
4. Use the first element of u as current input and repeat the whole process at the next step.

The optimised control value can then be applied to plant as shown in Fig.3.13. However, using the control variable to train a neural network controller has also been suggested [Hunt and Sbarbaro, 1992]. Training neural network controller provides two important options:

- To generate a useful starting value for the optimisation algorithm.
- To act as a stand-alone feedback controller without recourse to the optimisation step. This approach is similar to training a plant inverse.

Mayne and Michalska (1990) show that this technique is capable of stabilising linear and non-linear systems.

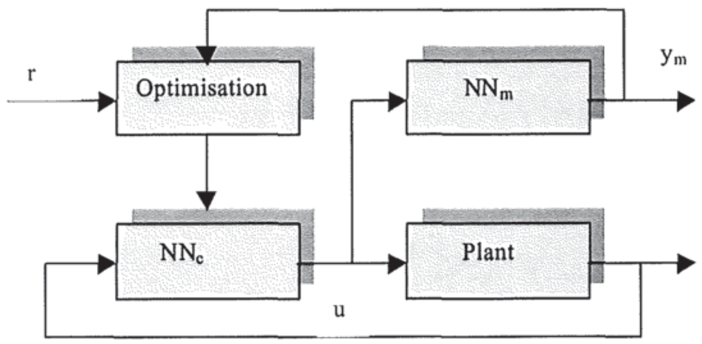


Fig. 3.13: Neural predictive control structure

3.5 Hybrid Strategy - Control Signal

3.5.1 Indirect learning architecture

The paper by [Morles and Mort, 1994] described a neural network-based control scheme where the neural network is trained in the indirect learning architecture. The control scheme also has a feedback controller, which is known as a filter *'G'* in the article (see Fig. 3.14). The tasks of the feedback controller are to compensate the effects that an unstable pole may have on the system response and to prevent having an improper ITO (Inverse transfer operator) [1] as a result of trying to control a strictly proper unknown dynamic system.

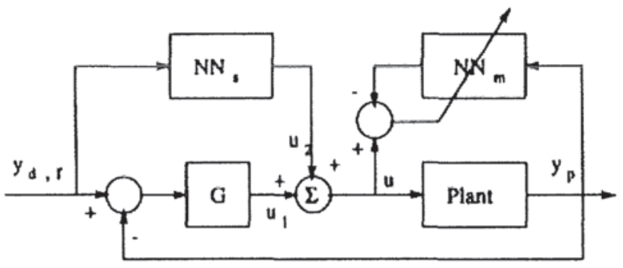


Fig. 3.14: Neural forward and inverse modelling with fixed controller

3.6 HYBRID STRATEGY – DESIRED OUTPUT SIGNAL

Two control schemes are presented here. The first control scheme is one proposed by [Lightbody and Irwin ,1995] and the second control scheme is suggested here.

3.6.1 Direct inverse control

Direct neural MRAC structure

Lightbody and Irwin (1995) proposed that neural network controller can be connected and used in parallel with a linear fixed gain controller. The fixed gain controller was first chosen to stabilise the plant and to provide approximate control. The plant can then be adequately driven over the operating range, with the neural network tuned on-line to improve the control. Fig. 3.15 shows the direct neural model reference adaptive controller (DNMRAC) structure. The terms K_r and K specify a nominal linear fixed gain state-space control law. The complete non-linear control provided by the network in parallel with a fixed gain state space controller is expressed as follows:

$$u(k) = K_r r(k) - K^T x_p(k) + u_{nn}(k)$$
$$u_{nn}(k) = NN[x_p(k), x_r(k), r(k)]$$

The neural network is trained using the system error:

$$e_s(k) = y_r(k) - y_p(k)$$

back propagated through the plant. The plant Jacobian is estimated through a neural network model as $J(k) \approx \frac{\partial y_m(k)}{\partial u(k-1)}$

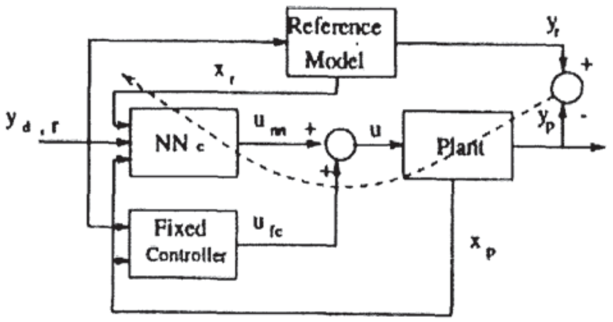


Fig. 3.15: Direct MRAC neural network

3.6.2 Forward modelling and inverse control

Song *et al.*, (1994) used the forward and inverse modelling technique, working together with a fixed controller described in Section 3.4.2. This control scheme is shown in Fig.3.16. The neural network controller is designed to overcome the non-linearities of the aero-engine. The fixed controller is utilised in the feedback loop to help the neural network training procedure and maintain engine stability. The fixed controller is designed based on Linear Quadratic Gaussian (LQG/ Loop Transfer Recovery (LTR) approach using information from the linearised model. The linearised model parameters can be provided by off-line identification. In their simulation example, the fixed controller is only used in the initial stage to guarantee the stability and safe operation of the aero-engine

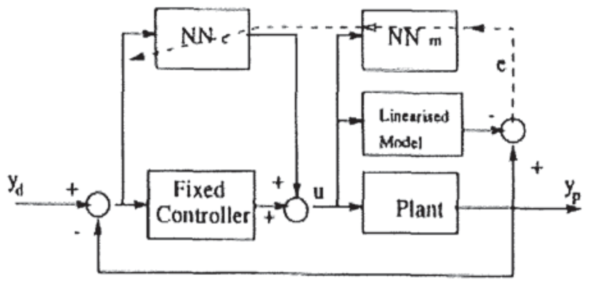


Fig.3.16 Neural forward and inverse modelling with fixed controller

If the control strategy of Song *et al.* (1994)'s is possible, then the following modification (Fig.3.17) may also be considered. The neural network controller is connected in a forward loop and the desired output is from a

[1] The ITO is estimated by the NN. It is actually an inverse modelling of the plant

reference model. The neural network input consists of the reference signal, past reference model and plant states. In this case, the neural network controller is acting like a feed-forward compensator.

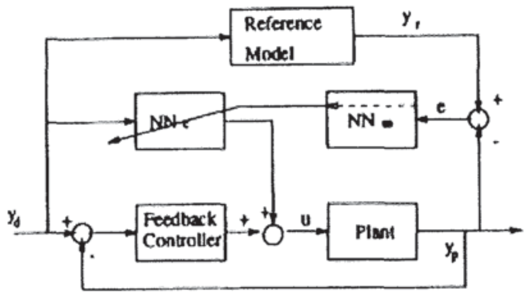


Fig. 3.17: Neural forward and inverse modelling with feedback controller

In [Omatu *et al.*, 1996], a neural network is trained in forward and inverse modelling before being applied in a hybrid structure to control an inverted pendulum. Fig.3.18 shows the hybrid structure proposed by [Omatu *et al.*, 1996]. Where it is claimed that the neural network controller is able to compensate offset errors due to non-linearities in the system.

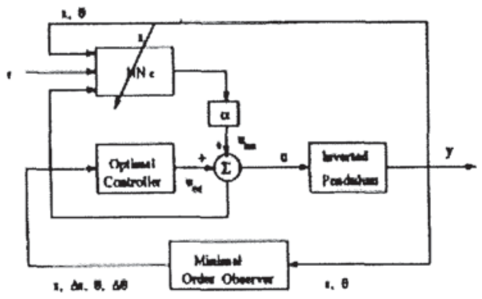


Fig. 3.18: Inverted pendulum system with neural network compensator

[Omatu *et al.*, 1996] also proposed a self-tuning PID type neural network controller as shown in Fig. 3.19. The neural networks are trained in the same forward and inverse modelling approach. However, the first neural network (NN_g) is used to determine the PID gains by minimising the cost function E defined as: $E = \frac{1}{2}e^2(k+1)$.

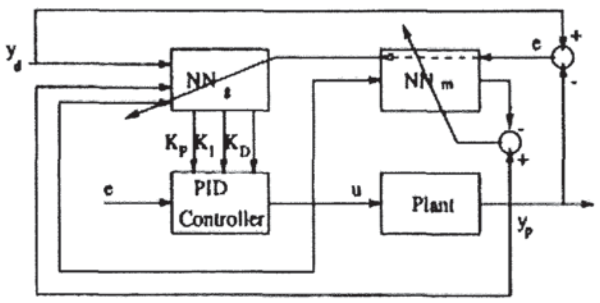


Fig.3.19: Self-tuning PID type neural network controller

3.6.3 Neural feedback linearisation

- Gaussian network for direct adaptive control

This architecture, proposed by Sanner and Slotine, 1992), employs a gaussian network or RBFN to compensate adaptively for plant non-linearities. This class of dynamic systems has the equations of motion expressed in canonical form as:

$$x^n(t) + f[x(t), \dot{x}(t), \ldots, x^{n-1}(t)] = gu(t) \qquad (3.13)$$

where $u(t)$ is the control input, f is an unknown non-linear function and g is the control gain (possibly state dependent). The structure of the proposed direct adaptive controller is shown in Fig.3.20.

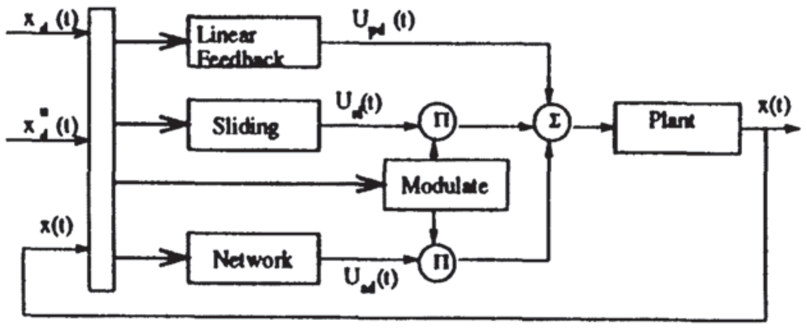


Fig. 3.20: Direct adaptive controller proposed by Slotine and Sanner

By integrating the adaptive and sliding components smoothly into a single control law, together with negative feedback of the tracking error states, it produces a globally stable strategy, which will asymptotically reduce the tracking errors to within the region of the origin. The control law used has the general structure as follows:

$$u(t) = u_{pd}(t) + 1 - m(t)u_{ad}(t) + m(t)u_{sl}(t) \qquad (3.14)$$

where $u_{pd}\ (t)$ is a negative feedback term including a weighted combination of the measured tracking error states, $u_{sl}(t)$ represents the sliding component of the control law, $u_{ad}\ (t)$ is the adaptive component and $m(t)$ is a continuous, state-dependent modulation which allows the controller to provide a smooth transition between sliding and adaptive modes of operation.

The sliding controller is present only as a backup system, meant to stabilize the system during the initial phases of learning. The sliding controller and modulation can be thought of as a formalization of the heuristic procedure, often used in neural network learning control applications, of monitoring and 'restarting' the learning process when a prespecified 'failure condition' is indicated.

3.7 Hybrid Strategy - Feedback Controller Output Signal

In this control strategy, the neural network is trained by using the feedback controller output signal. The most popular control structure using this strategy is feedback error learning.

3.7.1 Feedback error learning

The feedback error learning (FEL) method was proposed by [Kawata *et al.*1988] to train a neural network to perform dynamical control of a robotic manipulator [Kawato *et al.*, 1987; Kwato *et al.*, 1988]. The idea is to combine an already available and tuned conventional feedback controller with neural networks acting as the feed-forward controller. The feedback controller should at least be good enough to stabilise the plant when used alone, but it does not need to be optimally tuned. This scheme has some conceptual similarity to the conventional control structure called feed-forward compensation. The neural network can be said to be a forward acting compensator, which is external to the closed loop of the basic system and accomplishes error reduction, by non-feedback means.

The aim is to adapt the neural network in order to minimise the tracking error, e_s, defined as difference between a reference and measured output from the plant. In order to achieve this, Kawato used the output of the FC as the neural network output error and is therefore called the method the feedback error learning. Using this method, the problem of back-propagating the control error through the plant is avoided. Furthermore, the neural network can be trained on-line and the training method is goal oriented. This is because, when the output tracking error is zero, the output from the feedback controller will also be zero (in reality if there an integral component in the feedback controller, its output can be a non-zero constant, which case a bias term at the linear output unit of the neural network can be used to cancel such a constant output). Fig. 3.21 shows the structure of the FEL method.

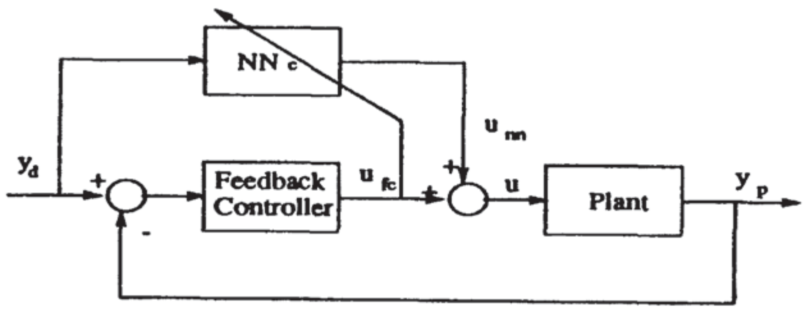


Fig. 3.21: Feedback error learning structure (FEL)

Gomi and [Kawato, l993] proposed two modifications of FEL for controlling robot arms applications. The first one is known as inverse dynamics model learning (IDML) where the neural network input is the plant output trajectory (acceleration, velocity and position). Hence, the neural network ultimately acquires an inverse dynamics model of the plant. The second modification is known as non-linear regulator learning (NRL). In this control scheme, the actual plant trajectory is not used as input to the neural network. Instead, the reference trajectory (acceleration, velocity and position) and the tracking errors less acceleration error are fed to the neural network.

3.8 Conclusions

This chapter has dealt with the following issues and are corner stone of understanding different types of methodology for neural network in control community:

- The discussion of various twofold classifications.
- Proposed multi-level categorization of various neural networks control strategies.
- Discussed and critical reviewed various neural network controllers.
- Suggested some possible control structures.

CHAPTER 4: NON-LINEAR SYSTEM IDENTIFICATION USING NEURAL NETWORKS

4.1 Introduction

System identification is the task of inferring a mathematical description, *a model*, of a dynamic system from a series of measurements on the system. There can be several motives for establishing mathematical descriptions of dynamic systems. Typical applications encompass simulation, prediction, fault detection and control system design. The interest in system identification goes far beyond the application to control hence it will be treated in a general fashion.

In this chapter the attention is drawn to

- identification of neural network models for non-linear dynamic systems. Apart from an increase in complexity compared to identification of linear systems, many of the results known from conventional system identification apply to neural-network-based identification as well. This includes a selection of material from conventional system identification, optimisation theory, nonlinear regression and the theory of neural networks.
- Network architectures including static and dynamic, single and multi-layer, and recurrent neural networks.
- The characteristic of neural network learning beginning from the very basic Hebbian, perceptron learning rule, Delta learning and the Widrow-Hoff rule.
- Special attention has been paid in this chapter is to develop most efficient learning algorithm for multi-layer networks; namely, BP learning in dynamic networks.
- Feedforward with inverse models, Internal model, Model reference, Predictive control
- Nonlinear self-tuning adaptive control

The topic of neural networks for identification and control is at present one of the active research areas in the field of control systems. Neural networks have been proposed by information and neural science as a result of the study of the mechanisms and structures of the brain. This has led to the development of new computational models, based on this biological background, for solving complex problems like pattern recognition [Tsypkin, 1971], fast information processing, learning and adaptation so on.

4.1.1 Historical background

In the early 1940s the pioneers of this [McCulloch and Pitts, 1943] studied the potential and capabilities of the interconnection of several basic components based on the model of a neuron. Others, like [Hebb, 1949], were concerned with the adaptation laws involved in neural systems. Rosenblatt coined the name Perceptron and devised an architecture, which has subsequently received much attention. Minsky and Papert introduced a rigorous analysis of the Perceptron. In the 1970s the work of Grosssberg came to prominence. This work, based on psychological evidence, proposed several architectures of non-linear dynamic systems with novel characteristics. Hopfield applied a particular dynamic structure to solve technical problems like optimisation. In 1986, the parallel-distributed processing group published a series of results and algorithms [Rummelhart *et al.*, 1986]. This work gave a strong impulse to the area and provided the catalyst for much of the subsequent research in this field. An excellent collection of the key papers in the development of the models of neural networks can be found in [Anderson and Rosenfeld, 1988]. Many examples of real world applications ranging from finance to aerospace are explored in [Hecht-Nielsen, 1989].

4.1.2 Neural network and control

In order to provide a rational assessment of new methods it is essential to compare the emerging technologies with well-established and traditional techniques. With reference to neural networks in control and identification, the following characteristics and features are important:

(a) Neural networks have the greatest promise in the area of non-linear control problems. This is implied by their theoretical ability to approximate arbitrary non-linear mappings.

(b) Neural networks have a parallel structure, which lends itself immediately to parallel implementation. Such an implementation can be expected to achieve a higher degree of fault tolerance and speed of operation than conventional schemes. Furthermore, the elementary processing unit in a neural network has a very simple structure. This also results in an increase of the processing speed.

(c) Hardware implementation is closely related to (b). Not only can networks be implemented in parallel, a number of vendors have recently introduced dedicated VLSI hardware implementations. This improves speed and increases the scale of networks, which can be implemented.

(d) As far as learning and adaptation is concerned, the networks are trained using past data records from the system under study. A suitably trained network has the ability to generalise when presented with inputs not appearing in the training data. Networks can also be adapted on-line.

(e) Data fusion. Neural networks can operate simultaneously on both numerical and symbolic data. In this, aspect networks stand somewhere in the middle ground between traditional engineering systems (numerical) and processing techniques from artificial intelligence (symbolic data).

(f) Neural networks have the ability to handle several inputs and several outputs, hence they are readily applicable to multivariable systems.

From the control systems viewpoint the ability of neural networks to deal with non-linear systems is perhaps most significant. The networks are used to provide the non-linear system models required by the techniques for synthesis of non-linear controllers. Neural networks based methods have an immense value for design of non-linear adaptive controllers for dynamic systems with poorly known and "difficult" dynamics. The learning algorithms are directly applicable as controller strategies for these controllers. Exactly the same arguments are applicable to non-linear identification, which is considered here as a component of a general area of control systems but not as a separate field (in contrast to a traditional setting). An excellent volume, which deals with such applications, is found in the IEEE Control Systems Magazine 1990. The compilation books [Miller *et al.*, 1990; Warwick *et al.*, 1995; Irwin *et al.*, 1995] provide a broad overview of the field of neural networks in control and identification.

4.2 Network Architecture

The network architecture is defined by the basic processing elements and the way in which they are interconnected (See Fig.4.1).

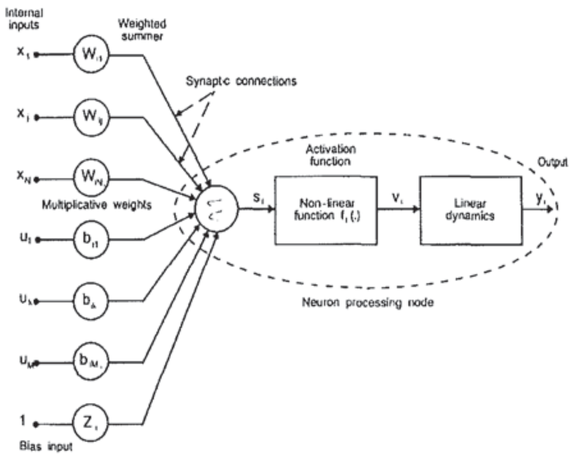


Fig.4.1: Basic processing node

4.2.1 Neurons (or processing element)

The basic processing element of the connections architecture is often called a neuron by analogy with neuro-physiology, but other names such as perceptron [Rosenblatt, 1958] or adaline [Widrow, 1989] are also used. The basic model of a neuron is illustrated in Fig.4.1. The neuron is composed of following three components:

(i) A weighted summer (ii) A linear dynamic SISO system and (iii) A non-dynamic, non-linear function, which is also, called the activation function.

The weighted summer is described by:

$$s_i(t) = \sum_{j=1}^{N_i} w_{ij} x_j(t) + \sum_{k=1}^{M_i} b_{ik} u_k(t) + z_i \tag{4.1}$$

giving a weighted sum (or net) in terms of the internal inputs x_i, external (control) inputs u_k and corresponding weights w_{ij} and b_{ik} together with constants z_i which play a role of standard bias; t denotes a time variable which can be either continuous or discrete.

Equation (4.1) can be written in a matrix form as:

$$s_i(t) = W_i x(t) + B_i u(t) + z_i \qquad (4.2)$$

The linear dynamic system has input v_i and output y_i. The variable y_i is the i^th neuron output. Its mathematical model can be written for continuous systems as:

$$T_i y_i(t) = v_i(t) \text{ (An inertia term)} \qquad (4.3)$$

or more generally as:

$$\alpha_0 y_i(t) + \alpha_1 y_i(t) = v_i(t) \qquad (4.4)$$

The discrete-time model can be represented as:

$$T_i y_i(t+1) + (1 - T_i) y_i(t) = v_i(t) \qquad (4.5)$$

The non-dynamic non-linear function $f_i(.)$ (activation function) gives the signal $v_i(t)$ in terms of the summer output

$$s_i(t) : v_i = f_i(s_i) \qquad (4.6)$$

There are different activation functions and their selection depends on the case under consideration. For example, in pattern recognition a threshold function is typically chosen, while in identification or control design the sigmoid and radial basis functions seem to be more popular (See Fig.2.2 in Chapter 2).

4.2.2 Static MFNNs

If $T_i = 0$ in (4.3) and (4.5) all neurons, then the network is static. The N static neurons in parallel fed by the input vector $x = (x_1,\ldots, x_L)^T$ and producing the output vector $y = (y_1,\ldots,y_n)^T$ constitute a one-layer feed-forward network which is illustrated in Fig. 4.2. The one layer network implements a static non-linear mapping relating the input x and the output y as (Fig. 4.3)

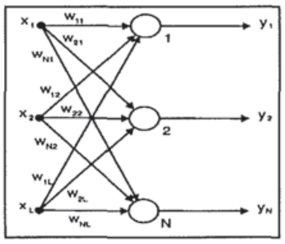


Fig.4.2: One layer feed-forward network

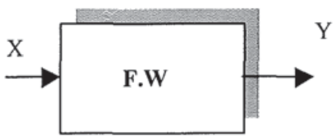


Fig.4.3: Non-linear mapping of a network

$$y = F(Wx) \qquad (4.7)$$

where

$$W = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1L} \\ \vdots & \vdots & \vdots & \vdots \\ W_{N1} & W_{N2} & & W_{NL} \end{bmatrix} \qquad (4.8)$$

and

$$F = diag(f_1(s_1,..., f_N(s_N)) \qquad (4.9)$$

Therefore, the network together with control inputs can be described with two mappings (linear and non-linear) as:

$$y = F(s) \text{ and}$$

$$s = Wx + Bu + z \quad \text{or} \qquad y = F(Wx + Bu + z) \qquad (4.10)$$

Connecting the three layers in a cascade, a two-layer feed-forward neural network is obtained which is illustrated in Fig. 4.4. The first layer is called the input layer and is composed of three neurons. The input layer output vector $y^1 = (y_1, y_2, y_3)^T$ is an input to the next layer, called the hidden layer, which is composed of two neurons. The third layer is called the output layer, produces the network output vector $y^3 = (y_6, y_7, y_8)^T$. The output layer is fed by the hidden layer output vector $y^2 = (y_4, y_5)^T$.

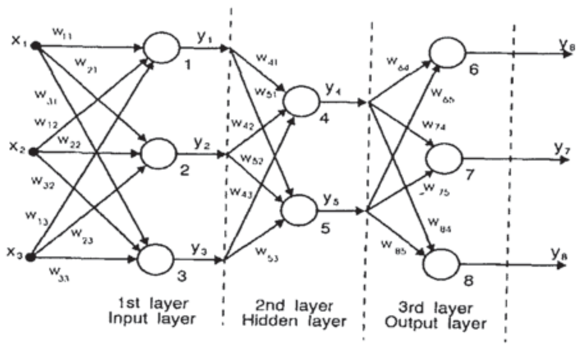


Fig. 4.4: Three-layer feed-forward neural network

The network implements a non-linear mapping relating input x and out y^3 in the following composed way:

$$y^3 = [F^3[W^3F^2[W^2F^1[W^1x]]]]$$

where F^1, F^2, F^3 and W^1, W^2, W^3 are the matrices of activation functions and weights of the layers, respectively.

4.2.3 Approximation properties of the FNNS.

It has recently been shown by [Hornik *et al.*, 1989] using the Stone-Weierstrass theorem, that a two layer network with a suitable number of nodes in the hidden layer can approximate any continuous function $h(.) \in C(R^L,R^N)$, over a compact subset of R^L.

This implies that feed-forward neural networks with even one hidden layer are adequate for purposes of characterisation. Since polynomials and orthogonal expansions can also approximate functions in $C(R^L, R^N)$ to any degree of accuracy, the advantages of neural networks over such representations are less obvious and have to be justified on the basis of practical considerations. In particular, the following questions have to be addressed if representations using neural networks are to be preferred:

Q1 Are neural networks a more efficient representation of special classes of continuous functions in that they need fewer parameters? If so, what are the characteristics of such functions?

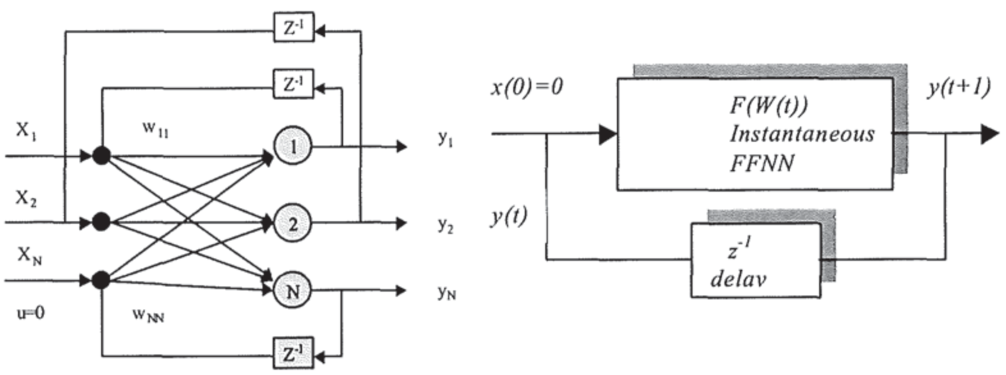
Q2 Given a non-linear mapping $h(\cdot)$ which has to be approximated, what dictates the choice of the number of layers and the number of nodes in each layer of the network?

These questions, which have received considerable attention, have only partial answers at present. However, the results obtained by [Albertini and Sontag, 1992; Slotine and Sanner, 1993] and are favourable for neural networks.

4.2.4 Feedback Networks

The recurrent network, based on the work of [Hopfield, 1982], has been used as a content-addressable memory and in optimisation problems. One version of the Hopfield network is shown in Fig. 4.5 and consists of a single-layer neural network in the forward path connected to a delay in the feedback path. The control input in Fig.4.6 is assumed equal to zero. The network represents a discrete-time dynamic system with the state vector x. The choice of weights determines the equilibrium states of this non-linear dynamic system and thus the specific equilibrium to which state trajectory converges depends upon the initial conditions $x_1(0),\dots,x_N(0)$.
The following holds:

$$y(t+1) = F(W(t), y(t)) \tag{4.11}$$



(a) Connection scheme (b) Block diagram

Fig.4.5 Single layer feedback network

A two-layer recurrent network is illustrated in Fig.4.6 again with $u \equiv 0$. A multi-layer recurrent network with control input u and the corresponding feed-forward neural network implementing a feed-forward gain is illustrated in Fig. 4.10. The network represents a non-linear controlled dynamic system.

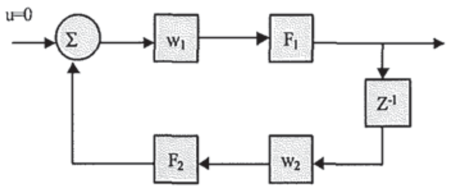


Fig.4.6: Two-layer re-current network

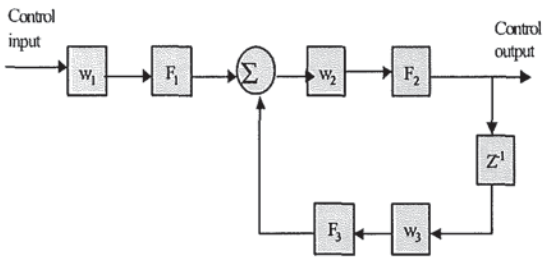


Fig.4.7: Multi-layer networks in dynamic systems

4.3 Learning in static networks

The classical formulation offered by approximation theory can be expressed as follows:
Given a continuous multivariable function $h(x)$ to be approximated by another function $H(x,w)$, where w is a vector of parameters. Let $\{x\}$ be a set of training examples of x. Find w^* such that:

$$\rho[H(x,w^*),h(x)] \leq \rho[H(x,w),h(x)] \tag{4.12}$$

For $x \in \{x\}$ and w, where $\rho(.,.)$ measures a distance between two functions at the training examples.

Typically, $\rho(.,.)$ is a sum of square differences taken for $x \in \{x\}$. The process of updating the weights in order to minimise ρ and obtain the best value w^* of the parameter w is called learning. The learning may be supervised or unsupervised as shown in Fig.4.8 and Fig.4.9.

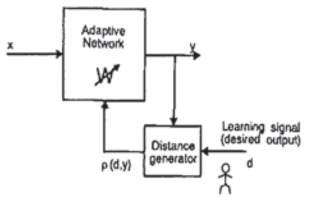


Fig 4.8: supervised learning

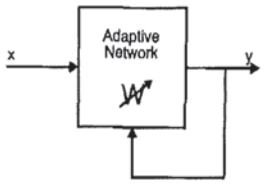


Fig 4.9: Unsupervised learning

4.3.1 Learning rules for a neuron

A general rule is based on work by [Amari, 1990] and is formulated as follows (See Fig.4.10):

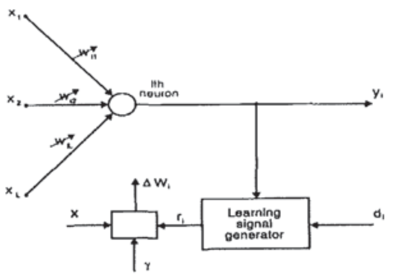


Fig.4.10: Illustration for weight learning (provided only for supervised learning) rules

The weight vector w_i increases in proportion to the product of input x and learning signal r_i.

The learning signal r_i is in general a function of w_i, x and sometimes of d_i the reference, or desired signal. hence:

$$r_i = r_i(w_i, x, d_i) \tag{4.13}$$

The increment of the weight vector is produced as:

$$\Delta w_i(t) = \gamma r_i[w_i(t), x(t), d_i(t)]x(t) \tag{4.14}$$

Where γ is a positive number called the learning constant that determines the rate of learning.

The weight vector adopted at time t becomes at the next instant, or learning step,

$$w_i(t+1) = w_i(t) + \gamma r_i\,[w_i(t), x(t), d_i(t)]x(t) \tag{4.15}$$

The superscript convention will be also used to index the discrete-time training steps as in the above equation. For the k^{th} step we thus have that

$$w_i^{k+1} = w_i^k + \gamma_i\,[w_i^k, x^k, d_i^k)x^k, k = 1,2,... \tag{4.16}$$

Continuous-time learning can be expressed as:

$$\frac{dw_i(t)}{dt} = \gamma r_i\,[w_i(t), x(t), d_i(t)]x(t) \tag{4.17}$$

4.3.1.1 Learning Rule [*Hebbian Rule,* 1949]

$$r_i \triangleq f(w_i^T x) = f_i(s_i) \tag{4.18}$$

and

$$\Delta w_i = \gamma f_i(s_i) \tag{4.19}$$

Thus, the single weight w_{ij} is adapted using

$$\Delta w_{ij} = \gamma f_i(s_i)x_j \tag{4.20}$$

This learning requires the weight initialisation at small random values around $w_i=0$ prior to learning. The Hebbian learning rule represents a purely feed -forward, unsupervised learning. The rule states that if the cross product of output and input or correlation term $f_i(s_i)x_j$ positive, this rules in an increase of weight w_{ij}, otherwise the weight decreases.

4.3.1.2 Perceptron Learning rule [*Rosenblatt,* 1958]

For the perceptron rule, the learning signal is the difference between the desired and actual neuron response. Thus, learning is supervised and the learning signal is equal to:

$$r_i = d_i - y_i \tag{4.21}$$

The zero mean threshold activation function (see Fig 2.2(b) in Chp.2) is used and, therefore,

$$y_i = sgn(s_i) = sgn(w_i^T x) \tag{4.22}$$

weight adjustments in this method are obtained as:

$$\Delta w_i = \gamma[d_i - sgn(w_i^T x)]x \tag{4.23}$$

and

$$\Delta w_{ij} = \gamma[d_i - sgn(w_i^T x)]x_j, \quad \text{for } j=1,2,\ldots,L. \tag{4.24}$$

because the neuron response is only binary, the (4.24) reduces to

$$\Delta w_i = \pm 2\gamma x \tag{4.25}$$

where a plus sign is applicable if $d_i = 1$, and $sgn\ (w^T, x) = -1$.

4.3.1.3 Delta Learning Rule [*Mc Clelland et al.,* 1986]

The delta rule is applicable only if an activation function is differentiable and in the supervised mode. The leading signal for this rule is called "delta" and is defined as:

$$r_i = \left[d_i - f_i(w_i^T x)\right] f_i'(w_i^T x) \tag{4.26}$$

This learning rule can be readily derived from the condition of least squared error (LSE) between y_i and d_i. Calculating the gradient vector w.r.t. w_i of the squared error defined as:

$$E = \frac{1}{2}(d_i - y_i)^2 \tag{4.27}$$

This is equivalent to

$$E = \frac{1}{2}[d_i - f_i(w_i^T x)]^2 \tag{4.28}$$

we obtain the error gradient vector value

$$\nabla E = -(d_i - y_i f_i'(w_i^T x)x \tag{4.29}$$

Since the minimisation of the error requires the weight changes to be in the negative gradient direction, we take

$$\nabla w_i = -\eta \nabla E \tag{4.30}$$

where η is a positive constant. We then obtain from (4.29) and (4.30)

$$\Delta w_i = \eta (d_i - y_i) f^{'}(s_i) x \tag{4.31}$$

4.3.1.4 *Widrow-Hoff* learning Rule

The Widrow-Hoff learning rule is applicable to the supervised training of neural networks. It is independent of the activation functions of neurons used since it minimises the squared error between the desired output value d_i and the neuron's activation value s_i. The learning signal for this rule is defined as:

$$r = d_i - s_i = d_i - w_i^T x \tag{4.32}$$

thus, the weighting vector increment under this learning rule is:

$$\Delta w_i = \gamma (d_i - s_i) x \tag{4.33}$$

This rule can be considered as a special case of the delta learning rule if the activation function is simply the identity function, that is if $f_i(s_i) = s_i$.

Speed of convergence and the convergence itself of the learning rule depends on the constant γ. To make the learning algorithm more reliable and efficient its adaptive version was proposed by [Widrow and Lehr, 1990] for identity activation function. The constant γ is now updated according to the rule:

$$\gamma(x) = \begin{cases} \dfrac{x^{'}}{x^T x}, & if \quad x^T x \neq 0, \\ 0, & if \quad x^T x \neq 0 \end{cases} \tag{4.34}$$

and the corresponding weight increment is

$$\Delta w_i = \begin{cases} \alpha \gamma(x)(d_i - y_i), & if \quad x^T x \neq 0, \\ 0, & otherwise, \end{cases} \tag{4.35}$$

where α is constant reduction factor.

Assume that $x \neq 0$ which implies $x^T x \neq 0$. The error dynamics for the adaline whose weights are adjusted by the above rule (which is now the delta rule) can be obtained as follows. Using (4.34) we have that at the k^{th} iteration

$$E_i^{k+1} \underline{\underline{\Delta}} d_i^{k+1} - y_i^{k+1} \quad \text{and}$$

$$E_i^{k+1} - E_i^k = y_i^k - y_i^{k+1} = -\left[(w^{k+1})^T - [(w^k)^T\right]x = -\left(\frac{\alpha E_i^k x^T}{x^T x}\right)x = -\alpha E_i^k$$

hence,

$$E_i^{k+1} (1-\alpha) E_i^k \tag{4.36}$$

From (4.36) the error E_i^k converges to 0 at a rate $(1-\alpha)$ if and only if $0 < \alpha < 2$.

A generalisation of the above rule to cover case of non-differentiable activation function has been proposed by [Sira-Ramirez and Zak, 1991]. *The rule is described as:*

$$w_i^{k+1} = \begin{cases} w_i^k + \dfrac{\alpha E_i^k}{x^T \theta(x)}, & \textit{if} \quad x^T \theta(x) \neq 0 \\ w_i^k, & \textit{otherewise} \end{cases} \tag{4.37}$$

where $0 < \alpha < 2$.

The error dynamic is described by the (4.36). The generalised rule dynamics is described by (4.37) becomes the previous one if the generator $\theta(.)$ is an identity operator, i.e., if $\theta(x) = x$. If on the other hand

$$\theta(x) = \begin{bmatrix} \operatorname{sgn}(x_1) \\ \vdots \\ \operatorname{sgn}(x_2) \end{bmatrix} \tag{4.38}$$

the adaptation algorithms allow non-differentiable activation functions to be considered then convergence is guaranteed.

4.3.2 Delta learning rule for MFNNs and BP training algorithm

This section is focused on a training algorithm applied to MFNNs. The algorithm is called the *error back-propagation training* algorithm, back-propagation (BP) for short.

The BP algorithm allows exponential acquisition of input-output mapping knowledge within multi-layer networks. Similarly, as in simple cases of the delta learning rule training presented earlier input patterns are submitted sequentially during back-propagation training. If a pattern is submitted and its classification or association is determined to be erroneous, the current least mean-square (LMS) classification error is reduced. We shall assume the error to be expressed as:

$$E = \frac{1}{2} \sum_{m=1}^{P} [d_m - y(x_m, w)]^T [d_m - y(x_m, w)] \tag{4.39}$$

where d^m denotes desired output corresponding to the input x^m, P is the patterns (d^n, x^m) and $y(x^m,w)$ denotes vector output of the network corresponding to the input x^m and weight matrix w. Often we take the mean (expected) value of the error E is taken if the training patterns are generated randomly from the training set or if there is a network output measurement error. The delta rule operates then as a stochastic approximation algorithm. Further simplification is obtained by assuming $P=1$ and a deterministic case.

During the association (or classification) phase, the trained neural network itself operates in a feed-forward manner. However, the weight adjustments force the learning rules to propagate exactly forward from the input layer.

To derive the BP algorithms consider the example of the network illustrated in Fig. 4.7 in section 4.2.2. The error can be written as:

$$E = \frac{1}{2}(d_6 - y_6)^2 + (d_7 - y_7)^2 + (d_8 - y_8)^2 \tag{4.40}$$

where d_6, d_7, d_8 denote the network desired outputs corresponding to the prescribed inputs x_1,x_2 and x_3.

The error is, therefore a function of the weights w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, w_{23} (input layers) and w_{64}, w_{65}, w_{74}, w_{84}, w_{85} (output layers). By applying gradient descent, the algorithm for adjusting the weights $W=\{w_{ij}\}$ in order to minimise $E(W)$ can be written as:

$$w_{ij}^{k+1} = w_{ij}^k - \gamma\frac{\partial E(w_{ij}^k)}{\partial w_{ij}}, \gamma > 0 \tag{4.41}$$

where k denotes an iteration number.

Consider a link between neurons 4 and 8 and the corresponding weight w_{84}. The weight w_{84} influences E indirectly according to the following chain:

$$w_{84} \to s_8 \to y_8 \to E \tag{4.42}$$

Therefore, by applying well-known chain rule we obtain that:

$$\frac{\partial E(w_{84})}{\partial w_{84}} = \frac{\partial E}{\partial y_8}\frac{\partial s_8}{\partial w_{84}} = -(d_8 - y_8)f_8'(s_8)y_4 \tag{4.43}$$

lets denote

$$\partial_8(x) \underline{\underline{\Delta}} (d_8 - y_8)f_8'(s_8(x)) \tag{4.44}$$

and call $\delta_8(x)$ an *equivalent error* associated with the eighth neuron (or output y_8) and corresponding to the input x and weights W. Notice that the equivalent error becomes just an error if the activation function $f_8(...)$ is an identity function. Therefore, the formula for adjusting the weight w_{84} can be written as:

$$w_{84}^{k+1} = w_{84}^{k1} + \gamma \partial_8 y_4 \tag{4.45}$$

Similarly

$$\begin{aligned} w_{74}^{k+1} &= w_{74}^{k1} + \gamma \partial_7 y_4, \\ w_{64}^{k+1} &= w_{64}^{k1} + \gamma \partial_7 y_4, \\ w_{85}^{k+1} &= w_{85}^{k1} + \gamma \partial_8 y_5, \\ w_{65}^{k+1} &= w_{65}^{k1} + \gamma \partial_6 y_5 \end{aligned} \tag{4.46}$$

hence, the output layer adjusted weights as a result of the *(k+1)*th interaction of the training algorithm can be computed by determining the equivalent errors associated with the network outputs and then by using the formulas (4.45) and (4.46).

Lets consider now the hidden layer and the link connecting neurons 1 and 4. The corresponding weight w_{41} is adjusted according to the error gradient descent as:

$$w_{41}^{k+1} = w_{41}^{k1} - \gamma \frac{\partial E(w_{41}^{k})}{\partial w_{41}}, \tag{4.47}$$

The weight w_{41} influences the error E indirectly according to the following chain:

$$\begin{tikzcd}
 & & & s_6 \arrow[r] & y_6 \arrow[dr] & \\
w_{41} \arrow[r] & s_4 \arrow[r] & y_4 \arrow[ur] \arrow[r] \arrow[dr] & s_7 \arrow[r] & y_7 \arrow[r] & E \\
 & & & s_8 \arrow[r] & y_8 \arrow[ur] &
\end{tikzcd}$$

let us denote a relationship between E and y_4 as $\tilde{E}\ (y_4)$. Applying me chain rule we obtain:

$$\frac{\partial E}{\partial w_{41}} = \frac{\partial \tilde{E}}{\partial y_4}\frac{\partial y_4}{\partial s_4}\frac{\partial s_4}{\partial w_{41}} = \frac{\partial \tilde{E}}{\partial y_4} f_4'(s_4) y_1 \tag{4.48}$$

The derivative $\dfrac{\partial \tilde{E}}{\partial y_4}$ can be computed as (see the above structure diagram)

$$\begin{aligned} \frac{\partial \tilde{E}}{\partial y_4} &= \frac{\partial E}{\partial y_6}\frac{\partial y_6}{\partial s_6}\frac{\partial s_6}{\partial w_4} + \frac{\partial E}{\partial y_7}\frac{\partial y_7}{\partial s_7}\frac{\partial s_7}{\partial y_4} + \frac{\partial E}{\partial y_8}\frac{\partial y_8}{\partial s_8}\frac{\partial s_8}{\partial y_4} \\ &= -(d_6 - y_6) f_6'(s_6) w_{64} - (d_7 - y_7) f_7'(s_7) w_{74} - (d_8 - y_8) f_8'(s_8) w_{84} \\ &= -\partial_7 w_{64} - \partial_7 w_{74} - \partial_8 w_{84} \end{aligned} \tag{4.49}$$

now define an equivalent error for the output of the fourth neuron as:

$$\partial_4 \underline{\underline{\Delta}} (\partial_6 w_{64} + \partial_7 w_{74} + \partial_8 w_{84}) f_4'(s_4) \tag{4.50}$$

Then, due to (4.48), (4.49) and (4.50) the following holds:

$$\frac{\partial E}{\partial w_{41}} = \partial_4 y_1 \tag{4.51}$$

and due to (4.47)

$$w_{41}^{k+1} = w_{41}^{k} + \gamma\partial_4 y_1 \tag{4.52}$$

Notice that the rule for adjusting the weight in the hidden layer has the same structure as the rule for adjusting the weights in the output layer. However, the equivalent error is now described by a more complicated formula (4.50). In this way the calculations have been organised to determine the hidden layer equivalent errors in a recursive way. The procedure, which is the merit of BP algorithms, starts from the output layer equivalent errors and propagates backwards along the network structure to the considered neuron.

The errors are multiplied by the corresponding weights and added. The resulting sum is then multiplied in a standard way by a derivative of the neuron activation function (4.50).

The BP algorithm is an elegant and effective computational tool for adjusting the network weights.

4.4 Learning in dynamic networks

Now consider dynamic neural networks and also more general dynamic systems containing static neural networks as their components. The objective is to present a suitable training algorithm if a task of the system is to follow a desired trajectory over time period $[t_0, t_f]$. For simplicity, discrete-time systems will be considered. The error is defined as:

$$E(t)\underline{\Delta}d(t) - y(t)\,, \qquad t = t_0, t_0+1, \ldots, t_f \tag{4.53}$$

where $y(t)$ denotes the system output at the instant t. The performance criterion is defined as:

$$J = \frac{1}{2}\sum_{t_0}^{t_f} E^T(t)E(t) \tag{4.54}$$

If different input-output training patterns are to be considered the summation in (4.54) must be extended to cover all these patterns as well.

The constant matrix of the weights W is to be found. Again, the gradient descant technique will be employed and a formula for gradient $\nabla J(W)$ will be found in a form of the BP algorithm.

4.4.1 Back-propagation through time (BPTT): [*Rummelhar and McClelland*, 1986]

A single-layer recurrent network will be investigated which can be described with the following state equations:

$$y(t+1) = \Gamma(y(t), u(t)) \tag{4.55}$$

where Γ is a non-linear mapping and u is a control input. Fig.4.11 presents a block diagram of a system, which can be described by (4.55).

The idea of BPTT to unfold the network through time, i.e. replace the one-layer recurrent network with a feed-forward one with t_f layers (Fig.4.12) represented by the same neural network modelling the mapping Γ.

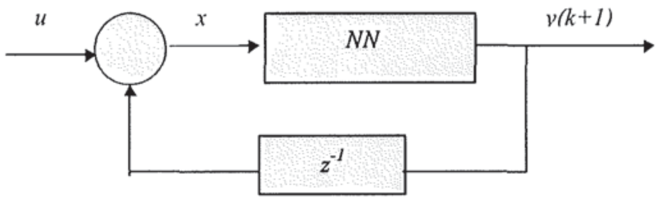


Fig.4.11: MFNN (z^{-1} is time shift operator)

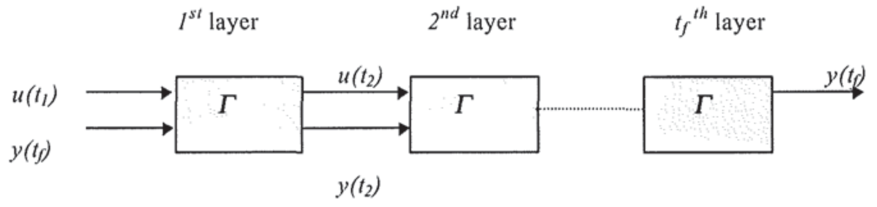


Fig.4.12: Structure of BPTT

It follows from (4.54) that:

$$\frac{\partial J(W)}{\partial w_i} = \sum_{t_0}^{t_f} E^T(t)\frac{\partial y(t)}{\partial w_i} \tag{4.56}$$

The derivatives $\frac{\partial E(t)}{\partial w_i}$ of the errors at subsequent time instants with respect to the weight w_i can be computed by applying a static BP scheme at each time instant based on the input produced of the previous time instant and the error corresponding to this time instant.

4.4.2 General dynamic back-propagation

Let us consider the system illustrated in Fig.4.11 and replace the time shift unit z^{-1} by a general time shift operation represented by t.f. $D(z)$. Notice, that

$$\frac{\partial E(t)}{\partial w_i} = -\frac{\partial y(t)}{\partial w_i} \tag{4.57}$$

The following holds:

$$y(t+1) = N(x(t+1), W),$$
$$x(t+1) = u(t) + D(z)y(k+1) \qquad (4.58)$$

Thus, and

$$y(t+1) = N(u(t+1) + D(z)\frac{\partial y(t+1)}{\partial w_i}, W) \qquad (4.59)$$

Hence,

$$\frac{\partial y(t+1)}{\partial w_i} = \frac{\partial N(x(t+1), W)}{\partial w_i} D(z)y(t+1) + \frac{\partial N(x(t+1), W)}{\partial w_i}) \qquad (4.60)$$

where arguments of all mappings and functions are taken at the normal values corresponding to current values of weights under applied control input.

The operator $D(z)y(t+1)$ is comparable to the output components at time instant t, $t-1$, $t-2, \ldots, t-n$. Therefore, (4.57) constitutes a recursive scheme, which can be used to evaluate the error derivatives at subsequent time instant (4.57). The derivatives $\frac{\partial N}{\partial x} and \frac{\partial N}{\partial w_i}$ must be computed separately and evaluated at every time instant.

The above approach can be applied to other dynamic systems including static neural networks and linear dynamics components.

4.5 Identification

The input and output of a time-invariant, causal discrete-time dynamic plant are $u(.)$ and $y_p(.)$, respectively, where $u(.)$ is a uniformly bounded function of time. The plant is assumed to be stable with a known parameterisation but with unknown values of the parameters. The objective is to construct a suitable NN identification model (Fig.4.13) which when subjected to the same input u as the plant, produces an output y^m which approximates y^p in a certain sense.

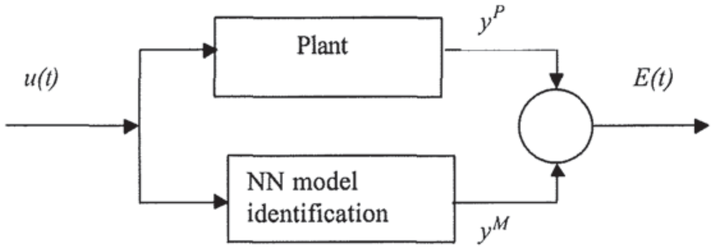


Fig.4.13: Identification

4.5.1 Forward modelling

The procedure of training a neural network to represent the forward dynamics of a plant will be referred to as forward modelling. The neural network model is placed in parallel with system and the error between the system and the network outputs (the prediction error) as the neural network-training signal. The BP training algorithm is applied to a MFNN.

Assuming that the plant is governed by a following non-linear discrete-time difference equation:

$$y^{p}(t+1)=F[y^{p}(t),\ldots,y^{p}(t-n+1);u(t),\ldots,u(t-m+1)] \qquad (4.61)$$

Thus, the plant output y^{P} at time $t+1$ depends on the past n output values and on the past m values of the input u. Concentrating, only on the dynamic part of the plant response such that the model does not explicitly represent plant disturbances. Special cases of the model (4.61) was introduced by [Narendra and Parthasarathy, 1990].

An obvious approach for system modelling is to choose the input output structure of the neural network to be the same as that of the system. Denoting the output of the network by y^{m} we then obtain that

$$y^{m}(t+1)=\hat{F}[y^{p}(t),\ldots,y^{p}(t-n+1);u(t),\ldots,u(t-m+1)] \qquad (4.62)$$

In the above, the mapping $\hat{F}(.)$ represents the non-linear input-output map of the network that approximates the plant mapping $F(\cdot)$. Note that the input to the network includes the past values of the plant output but not the past values of the network output (the network has no feedback). The learning static BP algorithm is used to find optimal values of the network weights. The structure of the model (4.62) is called *series-parallel*. The resulting identification structure is illustrated in Fig.4.13.

If after a suitable training period the network gives a good representation of the plant (i.e., $y^{m} \sim y^{P}$), then for subsequent post-training purposes the network output itself and its delayed values can be fed-back and used as part of the network input. In this way, the network can be used independently of the plant. Such a network is described as:

$$y^{m}(t+1)=\hat{F}[y^{m}(t),\ldots,y^{m}(t-n+1);u(t),\ldots,u(t-m+1)] \qquad (4.63)$$

This structure may also be used during the whole process of learning. The structure of (4.63) is called *parallel*. It may be preferred when dealing with noisy systems since it avoids the problem of bias caused by noise on the plant output. On the other hand, the series-parallel scheme (Fig.4.14) is supported by stability results. Moreover, the parallel model requires a dynamic back-propagation-training algorithm. Now consider a special case of (4.61):

$$y^p(t+1) = F[y^p(t),..., y^p(t-n+1); G[u(t),..., u(t-m+1)] \qquad (4.64)$$

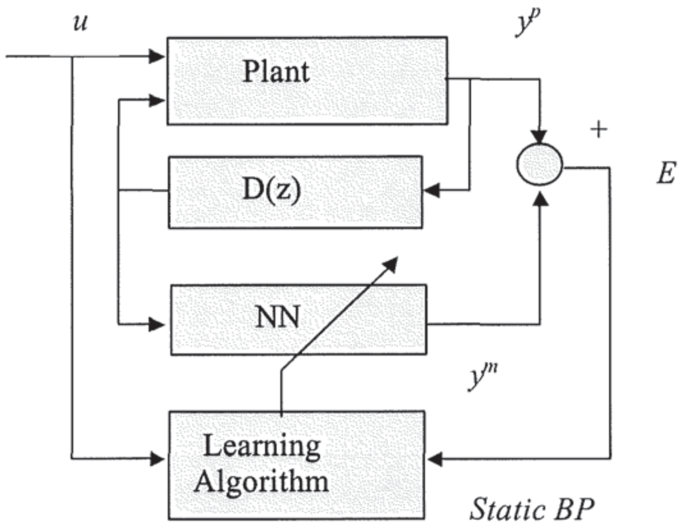


Fig.4.14: Series-parallel identification structure

Thus, the effects of the input and output values are additive. The structure of (4.64) is illustrated in Fig 4.18. Clearly, a general approach can be applied as presented before. However, it is reasonable to utilise the additive feature of the plant structure. Therefore, the model is described by the following series-parallel equations:

$$y^m(t+1) = \hat{F}[y(t),..., y(t-n+1)]; \hat{G}[u(t),..., u(t-m+1)] \qquad (4{\cdot}65)$$

where the mapping $\hat{F}$ and $\hat{G}$ are implemented by using two separate neural networks. The neural networks weights W_F and W_G are adjusted independently by static BP algorithm in order to achieve the best approximation of mappings $\hat{F}$ and $\hat{G}$.

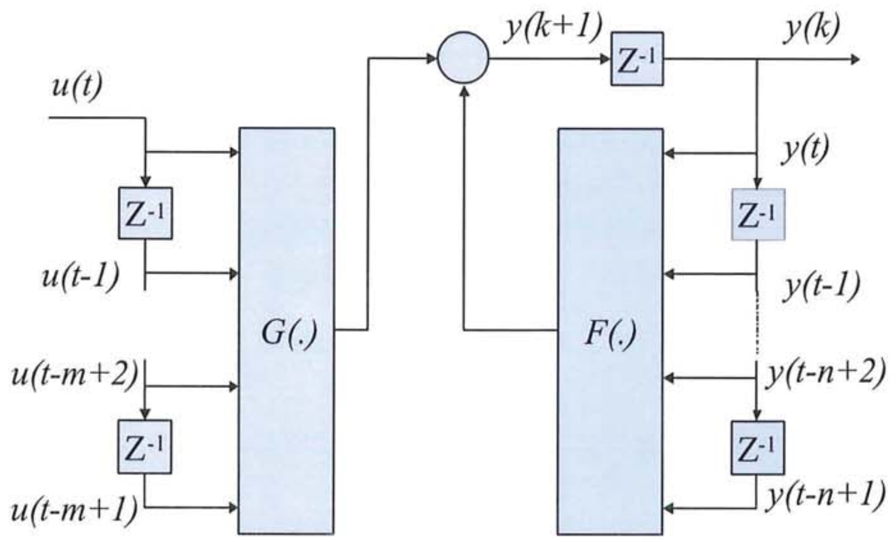


Fig.4.15: Structure of the plant dynamics

The importance of the class of inputs to be used to train learning systems is generally acknowledged. The training set has to be representative of the entire class of inputs that the system may be subjected to. This will ensure that the system will respond in the desired fashion even when subjected to inputs not included in the training data. This concept, referred to as persistent excitation, has been extensively treated in conventional control theory both in the context of identification and control problems by for example [Astrom and Wittenmark, 1989]. The concept of persistent excitation is also found to be important while dealing with the identification and control of non-linear systems using neural networks [Hunt *et al.*, 1992].

4.5.2 Inverse modelling

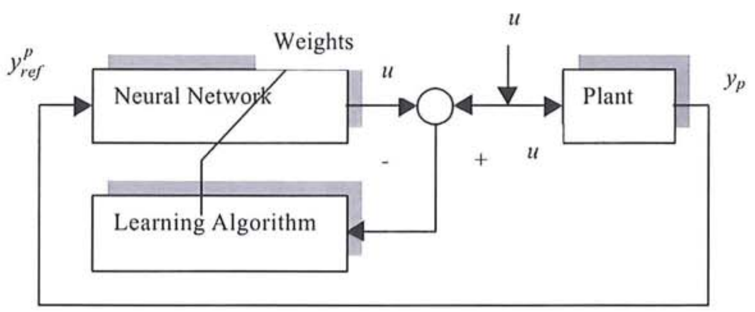


Fig.4.16: Direct inverse modelling

The inverse model of dynamic systems yields input for given output. The inverse models play a crucial role in a range of control structures. Some of the structures will be presented in the next section. However, obtaining inverse models raises several important issues [Hunt *et al.*1992]. Conceptually the simplest approach is direct inverse modelling as shown in Fig.4.16. Here, a synthetic training signal (the plant input) is introduced to the system. The system output is then used as input to the network. The network output is compared with the training signal (the system input) and this error is used to train the network. This structure will clearly force the network to represent the inverse of the plant. However, there are some drawbacks, which are as follows [Hunt *et al.,* 1992]:

- the learning procedure is not "goal directed" and the training signal must be chosen to sample over a wide range of system inputs and the actual operational inputs may be hard to define *a priori*. The actual goal in the control context is to make the system output behave in a desired way and thus the training signal in direct inverse modelling does not correspond to the explicit goal;
- if the non-linear system mapping is not one-to-one, then an incorrect inverse can be produced.

The first point is strongly related with the general concept of persistent excitation; the importance of inputs used to train learning systems is widely appreciated. Conditions for ensuring persistent excitation, which will result in parameter convergence are well established [Astrom and Wittenmark, 1989] For neural networks methods of characterising persistent excitation are highly desirable [Hunt *et al.*, 1992].

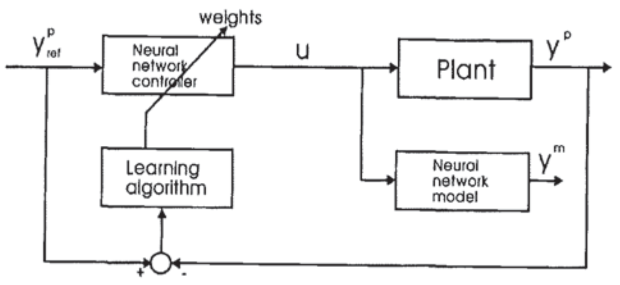


Fig.4.17: Specialised inverse modelling

A second approach to inverse modelling which aims to overcome these problems is known as specialised inverse learning [Psaltis & Yamamura, 1988]. The specialised inverse learning structure is shown in Fig.4.17. In this approach the network inverse model precedes the system and receives as input a training signal which spans the desired operational output space of the controlled system (*i.e.,* it corresponds to the system reference signal). This learning structure also contains a trained forward model of the system (e.g. a network trained as described in the Section 5.1) placed in parallel with the plant. The error signal for the training algorithm in this case is the difference between the training signal and the system output. It may also be the difference between the training signal and the forward model output if the system is noisy. It can be shown that using the plant output we can produce an exact inverse even when the forward model is not exact; this is not the case when the forward model output is used. The error may then be propagated back through the forward model and then the inverse model; only the inverse network model weights are adjusted during this procedure. Thus, the procedure is effectively directed at learning and identity mapping across the inverse model and the forward model. The inverse model is learned as a side effect.

In comparison with direct inverse modelling, the specialised inverse learning approach possesses the following features:

- The procedure is goal directed since it is based on the error between desired system outputs and actual outputs. In other words, the system receives inputs during training which correspond to the actual operational inputs it will subsequently receive.
- If the system forward mapping is not one-to-one, a particular inverse (pseudo-inverse) will be found. The problem of bias can also be handled.

Let us consider the input-output structure of network modelling the system inverse. From (4.61) the inverse F^{1} leading to the generation of $u(t)$ would require knowledge of the future value $y^{P}(t+1)$. To overcome this problem the future value is replaced with the value $y^{p}_{ref}(t+1)$ which is assumed available at time t. This seems to be a reasonable assumption since y^{p}_{ref} is typically related to the reference signal which is normally known one step ahead. Thus, the non-linear input-output mapping relation of the network modelling the plant inverse is:

$$u(t) = F^{-1}[y^{p}(t),\ldots,y^{p}(t-n+1); y^{p}_{ref}(t+1); u(t-1),\ldots,u(t-m+1)] \qquad (4.66)$$

that is the inverse model network receives as inputs the current and past system outputs, the training (reference) signal and the past values of the system outputs. Where it is desirable to train the inverse without the plant (see section above) the values of y^{P} in the above relation are simply replaced by the forward model outputs y^{m}.

4.6 Control

Models of dynamic systems and their inverses have immediate use for control purpose. In the control literature a number of well-established and analysed structures for the control of non-linear systems exist. For this study structures that have a direct reliance on system forward and inverse models will be focused upon. It is assumed that such models are available in the form of neural networks, which have been trained using the techniques outlined above. It is beyond the scope of this work to provide a full survey of all neural network based architecture available in the literature. The study concentrated upon presenting key directions and descriptions of the representative control structures.

4.6.1 Model reference control [Narendra and Parathasarathy, 1990]

The desired performance of the closed-loop system is specified through a stable reference model M, which is defined by its input-output pair $\{r(t), y^{r}(t)\}$. The control system attempts to make the plant output $y^{P}(t)$ match the reference model output asymptotically, *i.e.*, $\lim_{t\to\infty}\|y_r(t) - y_p(t)\| \leq \in$ for some specified constant $\in \geq 0$.

The model reference control structure for non-linear systems utilising the connectionist model is illustrated in Fig.4.18.

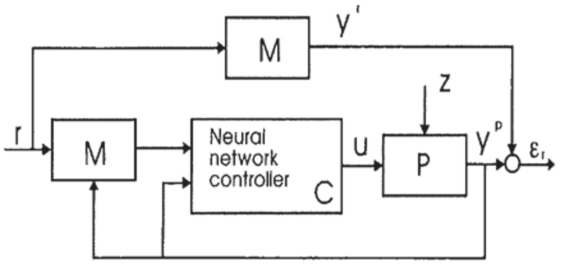


Fig.4.18: Model reference structure

In this structure the error defined above is used to train the network acting as the controller. This approach is related to training of an inverse plant model as presented in the previous section.

In general, the training procedure will force the controller to be a "detuned" inverse, in a sense defined by the reference model. The overall scheme can be viewed as direct adaptive control.

4.6.2 Internal model control (IMC)

In IMC, the role of system forward and inverse models is emphasised in [Gracia and Morari, 1982]. In this structure, system forward and inverse models are used directly as elements within the feedback loop. IMC has been thoroughly examined with the application of standard robustness and stability analysis. Moreover, IMC extends reality to non-linear systems control. A system model is placed in parallel with the real system. The difference between the system and model outputs is used for feedback purposes. This feedback signal is then processed by a controller subsystem in the forward path; the properties of IMC dictate that this part of the controller should be related to the system inverse. Given a network model for the system forward and inverse dynamics, the realisation of IMC using neural networks is straightforward [Hunt & Sbarbaro, 1991]. The system model *M* and the controller *C* (the inverse model) are realised using the neural networks as illustrated in Fig.4.19.

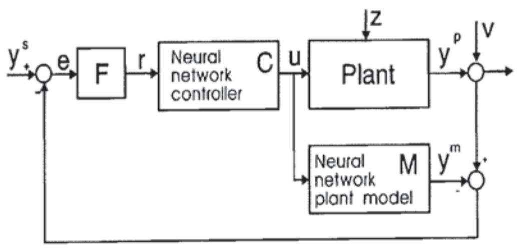


Fig.4.19: Internal model control structure

The subsystem F is typically a linear filter, which is designed to provide the desirable robustness and tracking response of the closed-loop system. It should be emphasised that the applicability of IMC is limited to open-loop stable systems.

4.6.3 Predictive control

In the realm of optimal and predictive control methods, the receding horizon technique has been introduced as a natural, computationally feasible feedback law. It has been proven that the method has the desired stability properties for non-linear systems.

In this approach, a neural network model provides a prediction of a plant's future response over a specified horizon. The predictions supplied by the network as passed to a numerical optimisation routine, which attempts to minimise a specified performance criterion in the calculation of a suitable control signal.

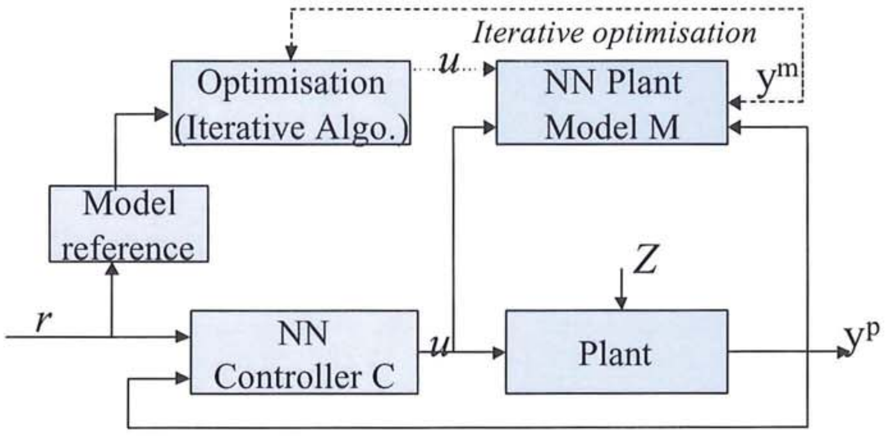


Fig.4.20: Neuro Model Predictive Control structure

The control signal $\tilde{u}$ is chosen to minimise the quadratic performance criterion, which compromises between the tracking error and the control cost [Clark and Gawthroup, 1975].

$$J = \sum_{j=N_1}^{N_2}[y^r(t+j) - y^m(t+j)]^2 + \sum_{j=1}^{N_u}\lambda_j[\tilde{u}(t+j-1) - \tilde{u}(t+j-2)]^2 \qquad (4.67)$$

Subject to the constraints and subject to the equality constraints introduced by the dynamic model itself.

Where, the constants N_1, N_2 and N_u define the minimum prediction (or cost), maximum prediction (or cost) and maximum control horizons respectively over which the tracking error and control increments are considered. $\lambda$is the control weight (or weighting factor penalizing changes in the control).

Once the iterative optimisation algorithm finds the optimal solution u it is applied to the plant. The actual value of the plant output y^p is measured and jointly with the reference signal r and y^p is sent to another neural network. This network is a controller and is trained to produce the same control output u as the optimisation routine. As the result, the non-linear feedback control law is obtained. An overall control structure is presented in Fig.4.20.

4.6.4 Self-learning controller [Nguyen and Widrow, 1990]

Consider a discrete-time dynamic system:

$$y(t+1)=F[y(t),u(t)] \tag{4.68}$$

where F is non-linear and unknown function.

A control problem is to provide the correct input vector $\{u(t)\}$ to drive the plant from an initial state to a subsequent desired state y^d. The objective is to design the state-feedback controller. We shall use a NN method for the controller and derive a suitable "learning" algorithm to adjust its weights. The resulting controller is non-linear adaptive. The control system structure is illustrated as in Fig.4.21.

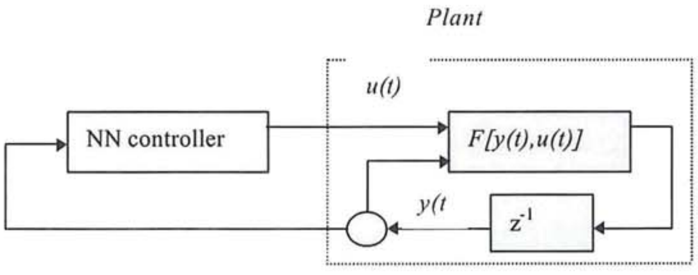


Fig.4.21: Control structure for the self-learning controller

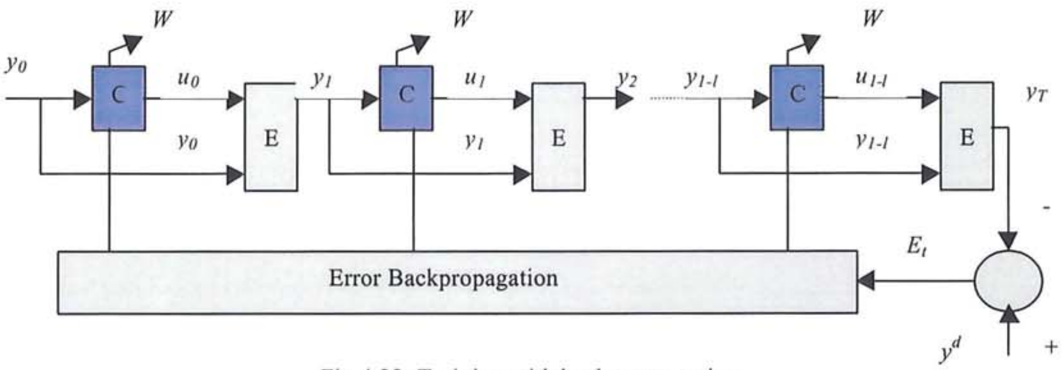


Fig.4.22: Training with back-propagation,

C- controller (another multi-layered NN, that learns to control the emulator)

E-plant emulator (a multilayer NN, that learns to identify the system's dynamic characteristics)

The learning algorithm requires the plant emulator (or competitor) which can be found in the form of NN by using the techniques presented in early section. The training procedure is as follows:

- The controller learns to drive the plant emulator from an initial state y_0 to the desired state y^d in T time steps.
- Learning takes place during many trials or runs, each starting from an initial state and terminating at a final state y_T.

The objective of learning process is to find a set of controller weights that minimises the error function, which is an average of $\left|y^d - yk\right\|^2$ over the set of initial states y_0. The training process is illustrated in Fig.4.22. The training starts with the NN plant emulator set in a random initial state y_0. Because the NN controller initially is untrained, it will produce an erroneous control signal u_0, to the plant emulator and plant itself. The plant emulator will then move to the next state $y_1 = y(1)$, this process continues for T time steps. At this point the plant is at the state $y_T = y(T)$. The designer should determine the number of time steps T. If weights are modified in the controller network the square error $\left|y^d - yk\right|^2$ will be less at the end of the next run.

To train the controller, the error in the controller output u_k for each time step k needs to be known. Unfortunately, only the error in the final plant state, y^d-yT is available. However, because the plant emulator is a neural network, we can propagate back the final error through the plant emulator using standard BP algorithm to get an equivalent error in the T^{th} stage. This error can be used to train the controller. The emulator, therefore, translates the error in the final Plant State to the error in the controller output. The real plant cannot be used here because the error cannot be propagated through it. This is why the NN emulator is needed.

The error continues to be BP scheme and the controller's weight change is computed for each stage. The weight changes from all the stages obtained from the BP algorithm are added together and then added to the controller's weights. This completes the training for one run. The overall scheme can be viewed as a self-learning controller. However, the controller parameter adjustment algorithm performs off-line and does not use the data from the real plant.

4.5.6 Non-linear self-tuning adaptive control

Consider a class of SISO feedback linearisable systems described by the discrete-time input *(u)*, output *(y)* equation:

$$y_{t+1} = F(y_t, y_{t-1}, \ldots, y_{t-p}, u_{t-1}, \ldots, u_{t-p}) + G(y_t, y_{t-1}, \ldots, y_{t-p}, u_{t-1}, \ldots, u_{t-p}) \qquad (4.69)$$

The functions *F(.)* and *G(.)* are unknown.

If we knew both F and G of (4.69), we could use the following control and the system would exactly track the desired output y^d_{t+1}:

$$u_t = -\frac{F(.)}{G(.)} + \frac{y_{t+1}^d}{G(.)} \tag{4.70}$$

Since F(.) and G(.) are unknown, NNs can be used to "learn" to approximate these functions and generate suitable controls. In order to simplify the problem and focus on the control mechanism, discussion will be limited to the 1st order system.

$$y_{t+1} = F(y_t) + G(y_t)u_t \tag{4.71}$$

Although the function *G(.)* is not known, we can assume that the sign of $G(y_t)$ is known along plant trajectories. The system (plant) can be modelled by the NN illustrated in Fig.4.23.

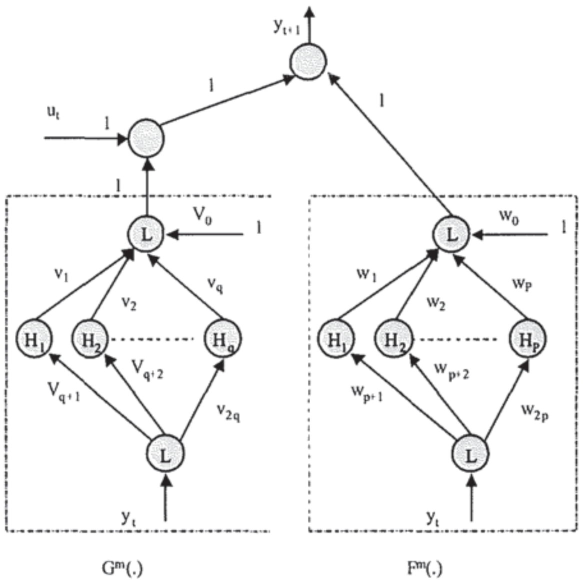


Fig.4.23: The neural network model

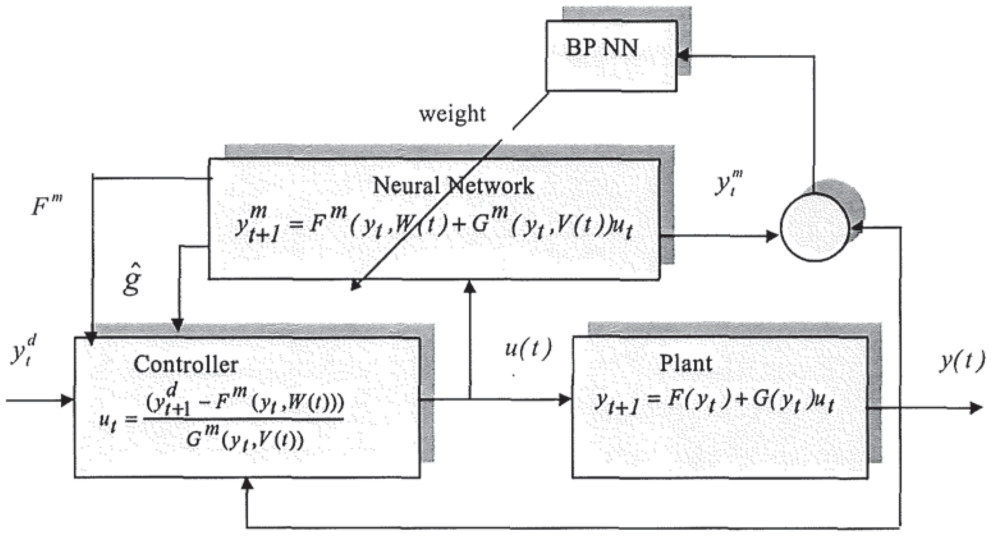
where $W = [w_0, w_1, .. w_{2p}]$ and $V = [v_0, v_1, .. v_{2q}]$ are the weights of the NNs approximating the function *F(.)* and *G(.)*, respectively (Fig.4.23). Therefore, the plant neural network model is described as:

$$y_{t+1}^m = F^m[y_t, W(t)] + G^m[y_t, V(t)]u_t \tag{4.72}$$

Notice that the model equation is series-parallel (See sec. 4.5). Notice also that $F^m(0,W) = w_0$ and $G^m(0,V) = v_0$. The linear neurons are labelled *"L"*. They are able to scale and shift incoming signals. The neurons labelled *"H"* are non-linear. Assume that there are enough neurons to approximate sufficiently accurately (with desired accuracy) the unknown function *F (.)* and *G (.)*. Although there exist in the literature general results relating the neuron number and the accuracy of approximation, in practice this number must be found heuristically.

The control task is for the plant output to track the command *{y_d (t)}*. The overall control system is illustrated in Fig.4.24. At time-step *t*, the following control (4.70) is applied to the plant (4.71):

$$u_t = \frac{F^m[y_t, W(t)]}{G^m[y_t, V(t)]} + \frac{y^d_{t+1}}{G^m[y_t, V(t)]} \tag{4.73}$$



BP NN
weight
Neural Network
$y^m_{t+1} = F^m(y_t, W(t)) + G^m(y_t, V(t))u_t$
y^m_t
F^m
$\hat{g}$
y^d_t
Controller
$u_t = \frac{(y^d_{t+1} - F^m(y_t, W(t)))}{G^m(y_t, V(t))}$
$u(t)$
Plant
$y_{t+1} = F(y_t) + G(y_t)u_t$
$y(t)$

Fig.4.24: Neural network self-tuning control system.

In contrast to "open-loop" training, the feedback dramatically changes the role of weights of the neural network. The output of the plant depends on the weights of the neural network and serves as the desired output of the neural network:

$$y_{t+1} = F(y_t) + G(y_t)\left[-\frac{F^m[y_t, W(t)]}{G^m[y_t, V(t)]} + \frac{y^d_{t+1}}{G^m[y_t, V(t)]}\right] \tag{4.74}$$

If (4.73) is substitute in to (4.72), the output of the neural network is y^d_t which is independent of *W(t)* and *V(t)*. Let us define the output error as:

$$E(t) = \frac{1}{2}(y^d_{t+1} - y_{t+1})^2 \tag{4.75}$$

Then, ***W(t)*** and ***V(t)*** are to be adjusted such that *E(t)* can be reduced. We can easily verify that:

$$\frac{\partial E(t)}{\partial w_i(t)} = \frac{G(y_t)}{G^m[y_t,V(t)]}\left[\frac{\partial F^m[y_t,W(t)]}{\partial w_i(t)}\right]E(t+1) \tag{4.76}$$

and

$$\frac{\partial E(t)}{\partial v_i(t)} = \frac{G(y_t)}{G^m[y_t,V(t)]}\left[\frac{\partial G^m[y_t,V(t)]}{\partial v_i(t)}\right]u_t E(t+1) \tag{4.77}$$

The quantities can be calculated by employing the standard BP algorithm.

$$\frac{\partial F^m[y_t,W(t)]}{\partial w_i(t)} \text{ and } \frac{\partial G^m[y_t,V(t)]}{\partial v_i(t)} \tag{4.78}$$

The quantity $G(y_t)$ is not known, but its sign is assumed to be known. Therefore, $G(y_t)$ is replaced by $sgn[G(y_t)]$ before (4.76) and 4.77) are used in the following gradient descant updating rules:

$$w_i(t+1) = w_i(t) - \eta_t \frac{\text{sgn}[G(y_t)]}{G^m[y_t,V(t)}\left[\frac{\partial F^m[y_t,W(t)]}{\partial w_i(t)}\right]E(t+1) \tag{4.79}$$

$$v_i(t+1) = v_i(t) - \mu_t \frac{\text{sgn}[G(y_t)]}{G^m[y_t,V(t)}\left[\frac{\partial G^m[y_t,V(t)]}{\partial v_i(t)}\right]u_t E(t+1) \tag{4.80}$$

The positive scalars η_t, μ_t specify the learning rates at time step t. With y_{t+1}, $W(t+1)$ and $V(t+1)$ available. (See 4.74, 4.79 and 4.80) the functions values $F^m[y_{t+1}, W(t+1)$ and $G^m[y_{t+1},, V(t+1)]$ are calculated by the neural networks. The $F^m[y_{t+1}, W(t+1)]$, $G^m[y_{t+1}, V(t+1)]$ and y^d_{t+2} can be used to generate u_{t+1} according to (4.73) for the next step.

The learning rates can be computed on-line based on current values of the plant output so that it can be shown by using Lyapunov theory that with these rates the tracking error converges to zero. It is possible to extend the scheme to more general situation.

The controller derived here uses on-line data from the plant to adjust its parameters and generate the control signal. It is a truly self-tuning non-linear controller.

4.7 SIMULATION RESULTS

In order to investigate the performance of the proposed algorithms described in early chapters the following examples are carried out. To illustrate the characteristics of the different controllers. In some cases simple system will be repeated in the examples in order to analyse the performance of the different types of controllers.

Software implementation

The structure of the network can be single, double or triple layered if more layers are to be incorporated by the user. Such a situation is common for example in BP through time. In principle, the network to be simulated is static multi-layer feed-forward. Introducing feedback can easily simulate recurrent networks.

BP algorithm modifications made are namely: the momentum method, improvement of initial conditions and adaptive learning rate, the examples provided could be easily customised to include the Levenberg-Marquardt learning rule as well as to use radial basis functions.

Off-line identification

The neural net models is not that trivial and efficient using the NN Toolbox, because all data constituting input and output plant measurements can be collected in matrices of so-called patterns. Learning is thus processing of the patterns, which is done using simple functions. Certain additional functions help in observation and evaluation of the progress of the learning as shown in the following examples. However, we should remember that for a large problem the convergence of learning could be too slow due to the way that MATLAB interprets its commands.

Off-line neural model identification

Off-line identification is performed by presentation of a predefined number of previously obtained patterns. The patterns are made up of a series of input and output signals. The approach is much the same as in standard identification (e.g. least squares) the only difference being that presentations of the data are repeated. One presentation is called an ***epoch*** (also see appendix A). The network consists of one hidden layer and has one output neuron related to model output $y(i)$. The number of input neurons follows from the assumed structure of the plant:

- NY : neurones related to delayed output of the plant. $y(i-1), y(i-2),..., y(i-NY)$
- NU+k : neurones related to the delayed input of the plant $u(i-1), y=u(i-k-2),..., u(i-k-NU)$

The number of hidden neurons *nun* are chosen by the user. The transfer function of the neuron can be sigmoidal (non-linear network) or linear (linear network).

Network structure

It is difficult to discuss the network structure choice if the plant is non-linear because is no general methods exist for non-linear system models.

Recursive (on-line) Identification

In adaptive control, the model has to be identified on-line in a recursive way. The following example provides concerning on-line identification of neural models. The structure of the neural model refers to the single layer discrete-time feedback network described in the main chapter. It is assumed that the discrete dynamical relation between input u and output y is:

$$y(i) = f(y(i-1), y(i-2),\dots y(i-nA),\ u(i-1-k), u(i-2-k),\dots u(i-nB-k))$$

where it is discrete time and f is an unknown, generally non-linear function. The number of input neurons is then $nA + nB$. k refers to additional discrete-time delay. One hidden layer consists of nun neurons. One output neuron refers to $y(i)$.

Example 1: On-line identification of linear neural model. The model has the form of linear feed-forward static neural network as shown in Fig.4.25 containing one hidden layer. The objective of this example to see how recursive identification of a linear model performs with neural model recursive identification.

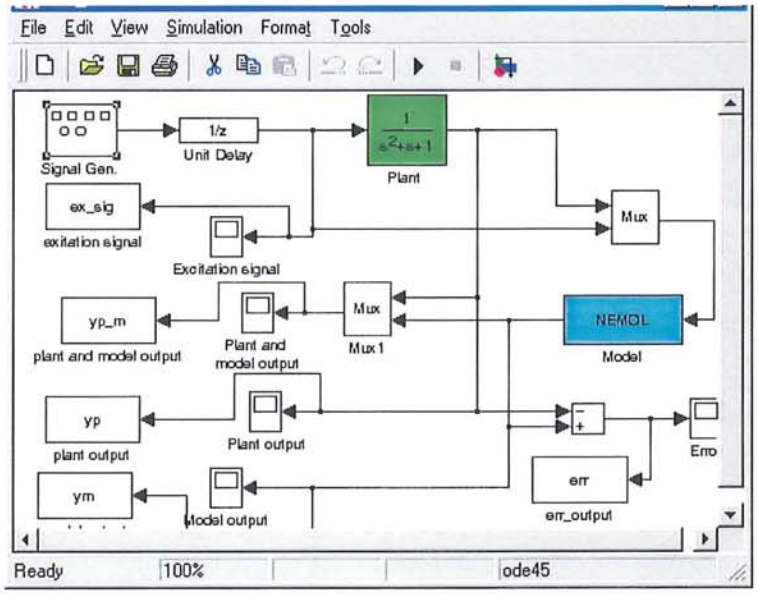


Fig.4.25: Block diagram of on-line identificatin of linear neural model

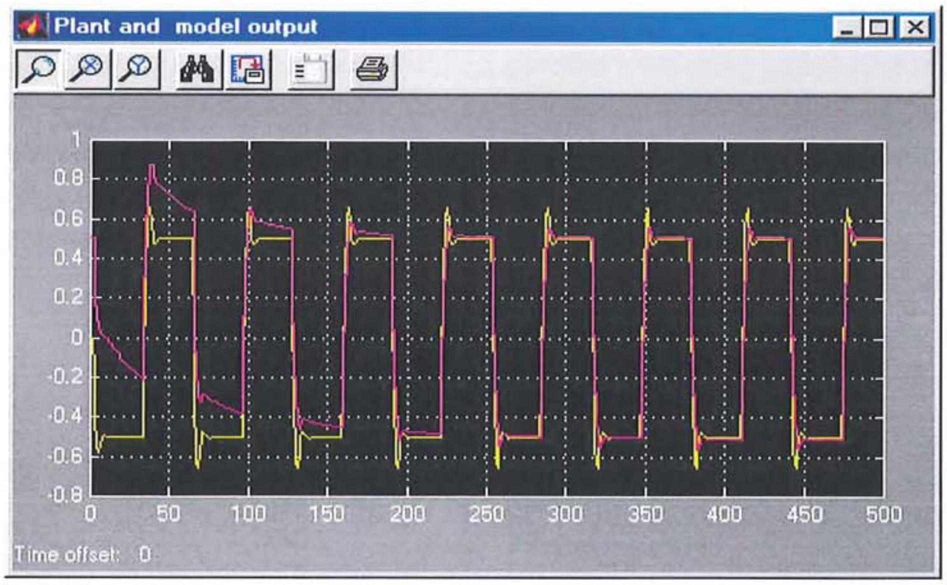


Fig.4.26: Linear plant (magenta) and model output (yellow) of example 1

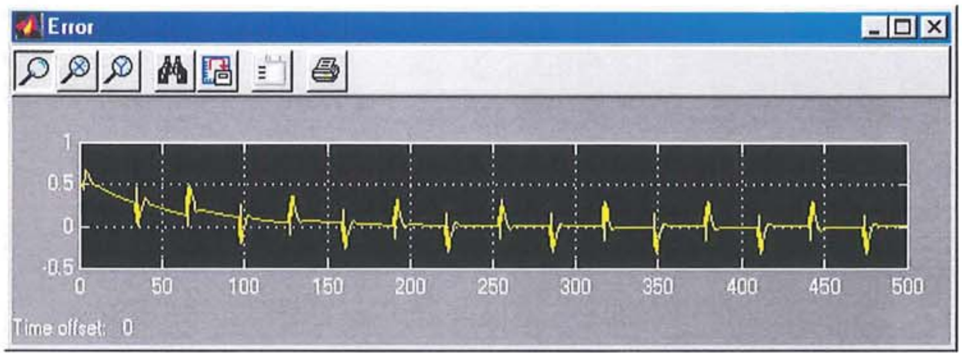


Fig.4.27: Error convergence of example 1

The parameters of the neural network model strutcure (NEMOL) are selected as [NY, NU, k, nun]= [1 1 0 5], Learning rate (lr) = 0.01, [offset, sampling time] =[0 1].

A square wave is applied as an input signal and a few simulation experiments are carried out with different amplitutes and frequencies of the input signal. The plant output and model output tracking response is as shown in Fig. 4.26. The convergence of identification shown is observed in Fig.4.27. The experiment was repeated with learning rate but it observed that it does not have any influance on tracking plant change. However, the improper setting of sampling time for the model and input signal sample and hold results in significant deterioration of on-line identification.

Example 2: Nonlinear online identification Fig.4.28 Block diagram for nonlinear on-line identification. The model has the form of sigmoidal feed-forward static neural network containing one hidden layer.

The nonlinear plant is of the form: $y(i)=\frac{y(i-1)}{1+y^2(i-1)}+u^3(i-1)$. This system was previously considered by [Narendara and Parthasarathy, 1990]:

Objectives

This example investigates how the constant learning rate is influenced by on-line nonlinear identification. The convergence and final approximation is observed (figure 4.31). This is in to identify the effect of learning rate when it is too high or low. In this way the difference between non-recursive (off-line) and recursive (online) when learning rate using the same epoch can be compared.

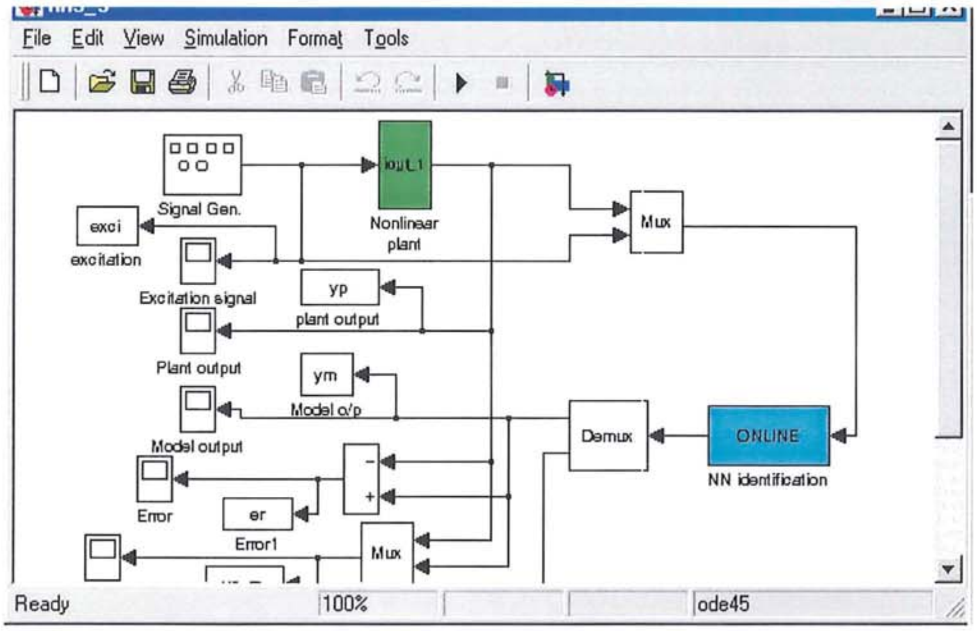
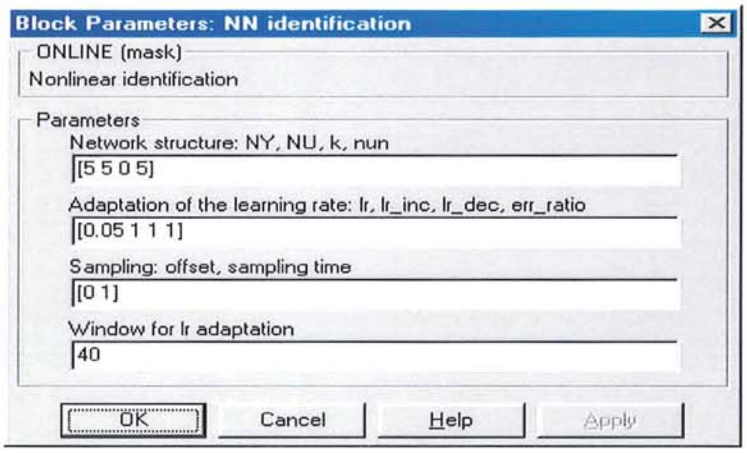


Fig.4.28: Block system diagarm of non-linear identification



ONLINE (mask)
Nonlinear identification

Parameters
Network structure: NY, NU, k, nun
[5 5 0 5]
Adaptation of the learning rate: lr, lr_inc, lr_dec, err_ratio
[0.05 1 1 1]
Sampling: offset, sampling time
[0 1]
Window for lr adaptation
40

Fig.4.29: Parameter dialogue box

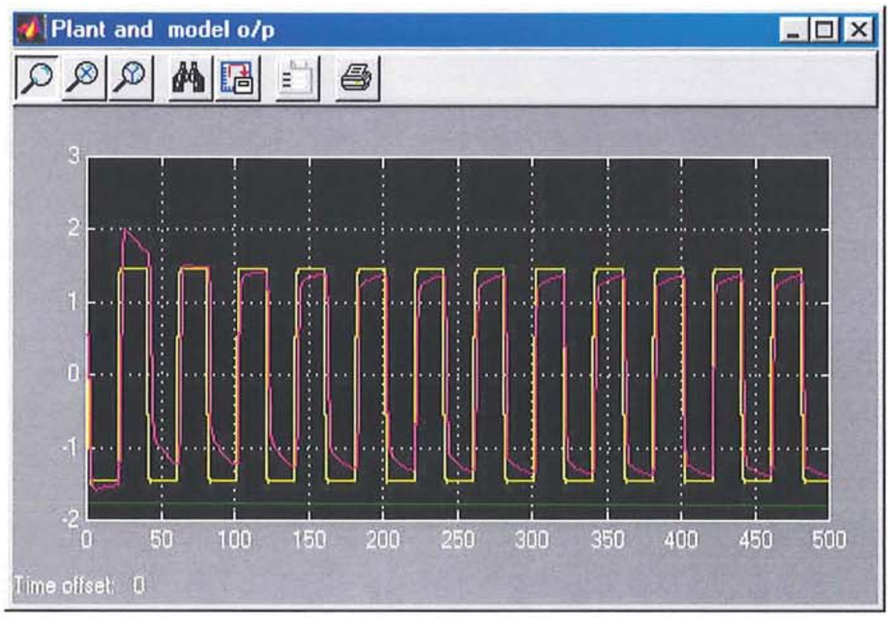


Fig.4.30 Plant (magenta) and model output approximation (yellow)

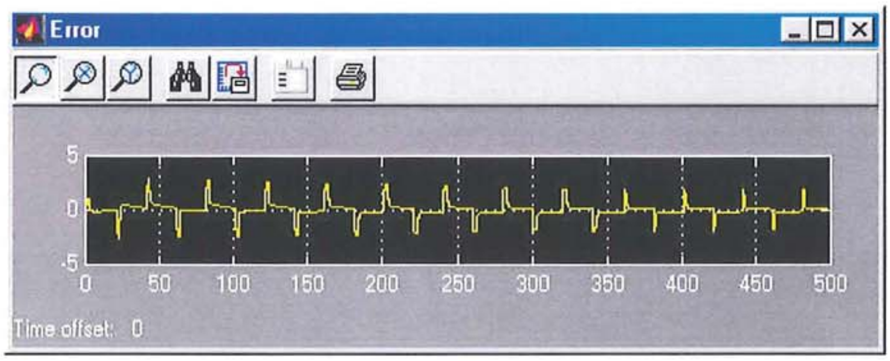


Fig. 4.31 Error convergence of example 2.

Figure 4.30 depicts the simulated response of the modeled plant. It demonstrates the convergence over time of the neural network, producing a close approximation of the plant response. This is also evident in the error convergence plot of figure 4.31.

The following plant dynamic systems are employed here to demonstrate the efficiency of the approximation and identification abilities of neural network controller and proposed identification structure. The plants are modelled using neural networks with 10 layers in both *F* and *G* networks as described below. The unknown non-linear function $p_1 * sin(p_2(y_k))$ is modelled by the neural network with two hidden layers. Each hidden layer contains n (n=10 used here) non-linear neurons. While one non-linear hidden layer is shown to be enough for approximating non-linear mapping, using more than one hidden layer may have the advantage of significantly reducing the number of hidden layer neurons needed for the same task. The initial weights, except $\hat{g}(0)$, and of the neural network are selected randomly between -0.1 to 0.1. The specific weight $\hat{g}(0)$ is chosen between

0.01 and 0.1 such that it has the same sign as the unknown positive number it is expected to approximate. Since the inverse of $\hat{g}(k)$ has a direct effect on the magnitude of u_k, $\hat{g}(k)$ is set to be 0.01 if $\hat{g}(k) < 0.01$. It is desirable to select small initial weights in order to avoid the saturation on non-linear neurons.

The learning rates μ and η are chosen to be 0.01 and 0.95 (see below) initially and decreased gradually later for better convergence. The choice of μ and η are made after observing the behaviour of the neural network for a long period of time and enough experienced is accumulated. When μ is too large, it is observed that $\hat{g}(k)$ jumps (burst) around the $\hat{f}[y(k), W(k)]$ and hardly learns to approximate $f(y_k)$.
So, it is important to select μ and η such that learning takes place at about the same pace in two subparts of the neural network, i.e., $\hat{g}(k)$ and $\hat{f}[y_k, W]$.

Example 3: The plant to be considered is described by: $y_{k+1} = p_1 \sin p_2(y_k) + p_3 u_k$

Objecives are:

- to learn how the dynamic behaviour of the nonlinear system model (Example 1) response when differenent excitation signals such as sin, square are applied
- to learn how an adaptive GPC controls the system
- to learn how a PI controller performs with nonlinear system
- to compare the quality of PI and GPC control
- to learn how a neural controller performs with the same nonlinear plant. The neural controller uses two networks 'f' and 'g' to describe two parts of the plant. Each network contains two hidden layers of neurons.
- To learn how neural predictive control with the same nonliner plant

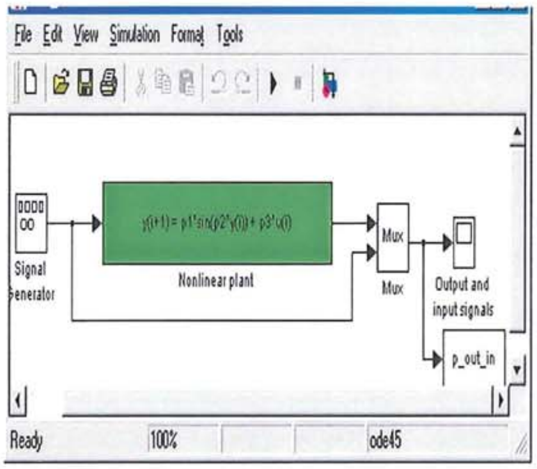


Fig.4.32: Block diagram for example 3 without controller

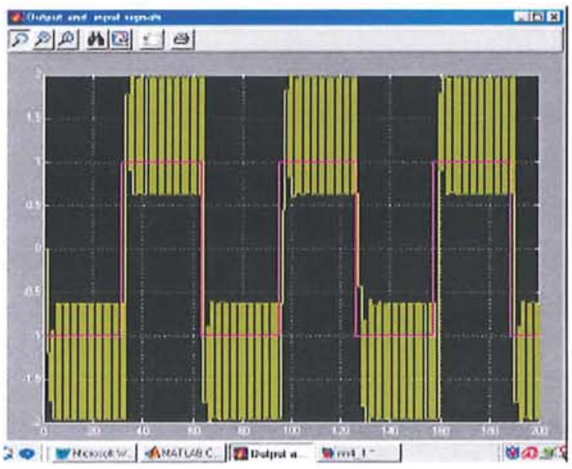
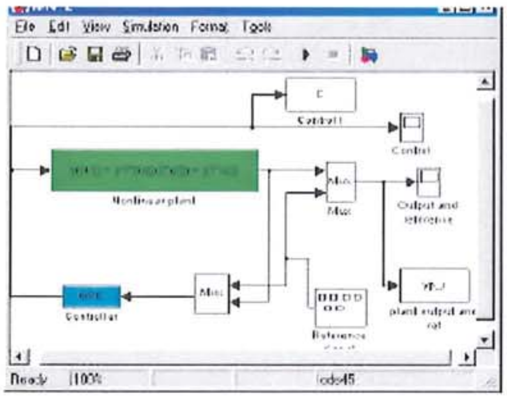


Fig.4.33: Representative simulation results example



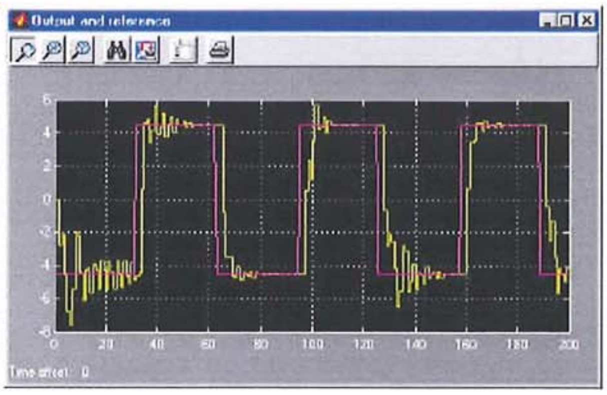
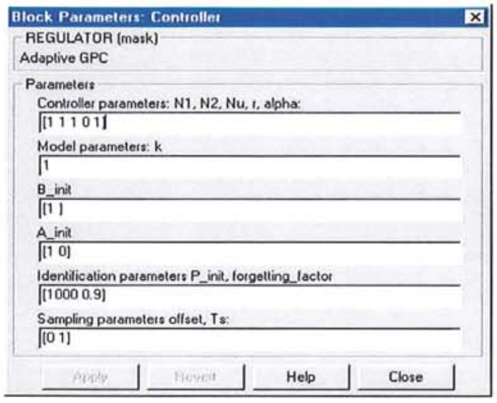


Fig.4.34: Block diagram with GPC controller Fig.4.35 Representative simulation with GPC



Block Parameters: Controller

REGULATOR (mask)

Adaptive GPC

Parameters

Controller parameters: N1, N2, Nu, r, alpha:
[1 1 1 0 1]

Model parameters: k
1

B_init
[1]

A_init
[1 0]

Identification parameters P_init, forgetting_factor
[1000 0.9]

Sampling parameters offset, Ts:
[0 1]

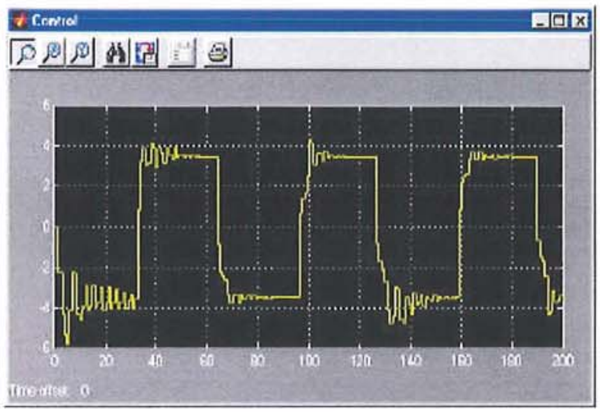
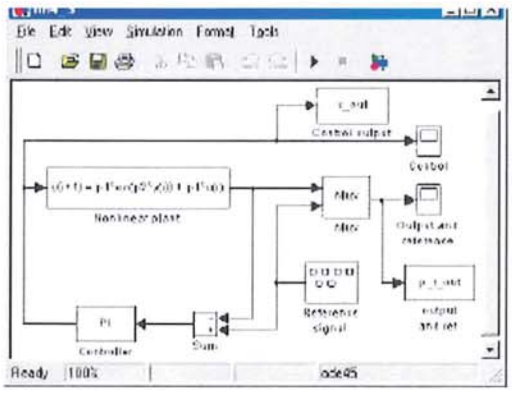


Fig.4.36: Parameter used in GPC controller Fig.4.37 Control ouput of GPC



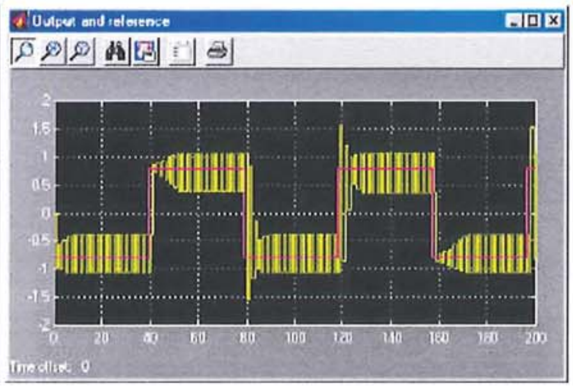
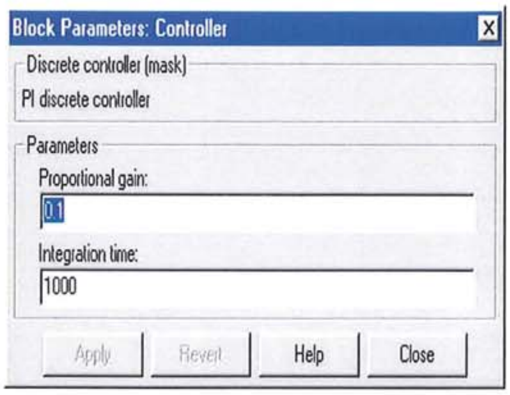


Fig.4.38: Block diagram with PI Fig.4.39 Output and refrence of PI



Block Parameters: Controller

Discrete controller (mask)

PI discrete controller

Parameters

Proportional gain:

0.1

Integration time:

1000

Fig.4.40: Parameter used in PI controller

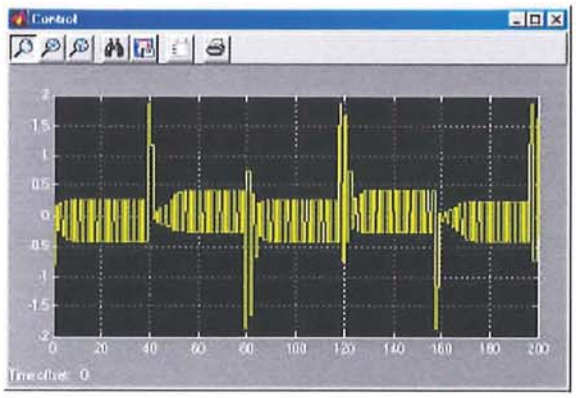


Fig.4.41 Output and refrence of PI

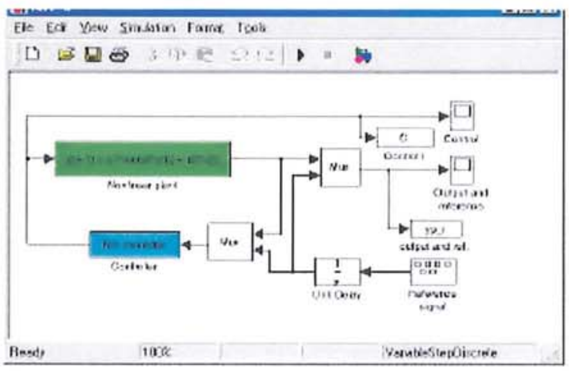


Fig.4.42: Block diagram with NN controller

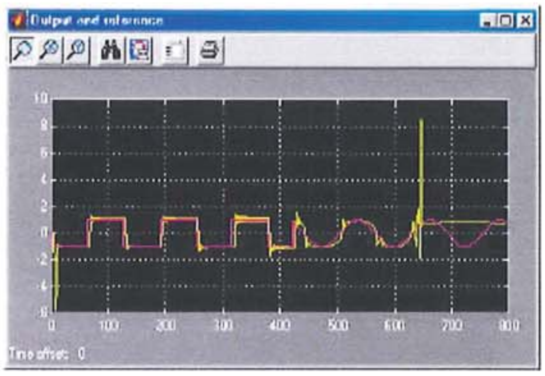
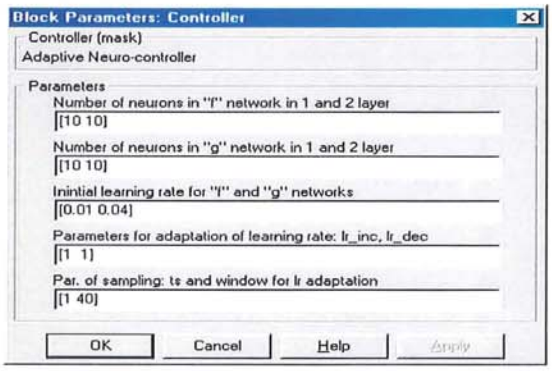


Fig.4.43 Output and refrence of NN



Block Parameters: Controller

Controller (mask)

Adaptive Neuro-controller

Parameters

Number of neurons in "f" network in 1 and 2 layer

[10 10]

Number of neurons in "g" network in 1 and 2 layer

[10 10]

Inintial learning rate for "f" and "g" networks

[0.01 0.04]

Parameters for adaptation of learning rate: lr_inc, lr_dec

[1 1]

Par. of sampling: ts and window for lr adaptation

[1 40]

Fig.4.44: Parameter used in NN controller

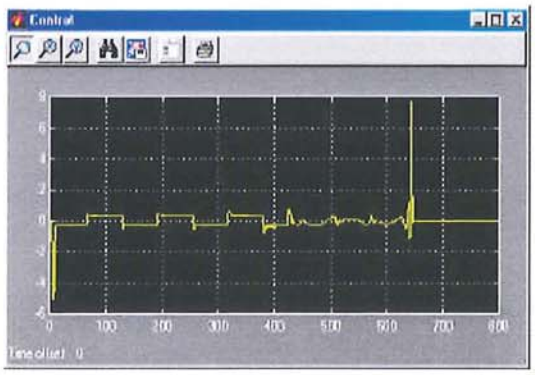


Fig.4.45 Control output of NN

The results of these simulations are shown in figure 4.32 through 4.45. Initially figure 4.32 and 4.33 show the uncontrolled dynamic plant response to which all subsequent controlled responses can be compared. Wild oscillation occur during peak cycles when a square wave is applied as the input signal. It is undesirable in practice, therefore each of the candidate adaptive controllers will be evaluated on their ability to compensate for this alongside their ability to control the plant to track the driving signals. Application of adaptive GPC contol to this system results in data presented in figures 4.34 to 4.37. These figures show a good peromance in terms of controlling plant oscillatory dynamics as well as tracking the demand.

The effect of PI controller on the system (Fig.4.41 through 4.44) shows that this type of controller is unable to adapt to the nonlinear system dynamic. Neural network reponse of figures 4.45 through 4.48 out-perform the other two candidate control methodologies under the criteria considered here. As a further demonstration of validity neural network controller the results data presented in figures 4.45 and 4.47 also include the case when the demand signal changes to a sinus mid simulation. These figure also demonstrate that the phenomenon of sudden burst. This can occur eventhough neural network controller performs well initially. This sudden burst could be due to unknown random error during the learning process.

Similar result but for different nonlinear plant also been reportd in previous paper [Thapa B. *et al.* 1999, 2000].

4.8 Conclusions

The work presented in this chapter is mostly concerned within a framework of identification and control problems. The various up-to-date known types of network architectures have been presented, such as static, dynamic, single, multi-layer and recurrent networks.

The most challenging tasks of great interest to scientists, engineers and others alike are to understand the characteristic of neural network learning. Hence various types of learning are presented such as a very basic Hebbian rule, perceptron learning rule, delta learning, Widrow-Hoff rule etc. Special attention has been paid in this chapter to the development of the most efficient learning algorithm for multi-layer networks; namely, BP learning in dynamic networks.

Learning algorithms are the basis for an introduction to identification problems. Forward and inverse modelling have been discussed and proper structure of identification systems has been presented. Neural models obtained via identification are intended to become a part of the control structure. Model reference control has been presented. The principle of neural inverse model technique has been applied. The scheme was characterised by the controller being a network trained as the inverse of the system. "Inverse" was understood in the sense that the transfer function for the closed-loop system, consisting of controller and system, equalled the time delay of the system. Two different methods for training the network were presented:

- **Generalised training** (off-line): In this scheme the network was trained off-line to minimize the mean square error between a control signal applied to the system in an initial experiment and the control signal produced by the neural network. The network could be trained with any of the training methods presented in Chapter early chapter.

- **Specialized training** (on-line): The objective is to minimize the mean square error between reference signal and the output of the system. This is done (on-line) with a recursive training algorithm. The issue of excitation is important in relation to generalized training. Due to the fashion in which the training set is collected, it is difficult to avoid that certain regimes of the operating range are not properly represented. Adding to this the problems with generating inverse models of systems not being one-to-one, it is concluded that generalized training should be used mainly for initializing the network. The network is subsequently fine-tuned with specialized training. A complete working procedure for standard problems is outlined in Table 4.1

Table 4.1: General procedure for training of inverse models

1.	Conduct an experiment to generate a data set.
2.	a "forward" model of the system is identified.
3.	Initialize the inverse model with general training.
4.	Proceed with specialized training "off-line" by using the model of the system instead of the actual system. Apply a recursive Gauss Newton algorithm with forgetting for rapid convergence but be careful with "covariance blow-up."
5.	Conclude the session by on-line specialized training. Terminate the training algorithm when an acceptable model-following behaviour has been achieved.

The major characteristics of direct inverse control are briefly recapitulated as follows:

Advantages

- Intuitively simple.
- Simple to implement.
- With specialized training the controller can be optimized for a specific reference trajectory.
- It is (in principle) straightforward to apply specialized training on time varying systems.

Disadvantages

- Does not work for systems with an unstable inverse, which unfortunately often occur when using a high sampling frequency.
- Problems for systems not being one-to-one (generalized training of the inverse models).
- Problems with inverse models that are not well damped (local linearized models will have zeros near the unit circle).
- Lack of tuning options.
- Generally expected to show a high sensitivity to disturbances and noise.
- Pretraining is needed and that limits its application to plants that can be pretrained.
- The neural network model needs to be accurately trained to work well. That would mean a long training

time is required.

Problems with biases may occur if the plant parameters change due to ageing or wear and tear of components

One of the most useful properties of neural networks is prediction ability hence predictive control is the structure in which this ability is incorporated in a special way. A prime characteristic that distinguish predictive control from other controllers, is the idea of a receding horizon (see Appendix D); at each sample the control signal is determined to achieve a desired behaviour in the following N_2 time steps. This idea is also appealing because it relates to many of the control tasks that one as a human being carries out on a daily basis. This intuitive foundation can to some extent accommodate the tuning of the design parameters.

The self-learning controller proposed by Nguyen and Widrow has been presented. Non-linear self-tuning adaptive control has been implemented and a simulation result shown that it is very promising for control of more complex non-linear plants.

The self-tuning adaptive control is traditionally limited to unknown linear systems. By introducing back-propagation neural networks into the self-tuning scheme, it is demonstrated that this new technique has the potential to deal with unknown linearizable non-linear systems.

COMMENTS

Comment 1: The identification of an unknown non-linear plant may be needed for off-line or on-line control. The nature of the identification in the two cases may be substantially different. The various considerations that determine the models to be used and the test signals to be applied need to be studied further.
Comment 2: In all the simulation studies, the output error tends to zero rapidly but the convergence of the weights cannot be assured. Concepts similar to persistent excitation encountered in adaptive systems theory are needed in this case also to assure convergence.
Comment 3: Control systems based on the results of the identification procedures are simple to implement. However they require a degree of engineering judgement in their application.
Comment 4: Computation of control Strategies and Implementation

- Can be quite time consuming in particular in the computations are made *off-line.* It might be necessary to repeat the procedure if the system dynamics or the characteristics of disturbances are changing, as is often the case for industrial processes.
- Integration of different techniques are the key role for understanding NN in control application.
- By introducing BPNNs into the self-tuning and identification control scheme, it is demonstrated that the new control method has the potential to deal with unknown feedback linearisable non-linear systems.
- The predictive control strategy applied here has many advantages; it is easy to tune flexible and provides a good controller performance.

- Behind the control scheme proposed here, many practical and theoretical questions could be raised.
- To derive a suitable non-linear model from a physical principle is in general much too complex for controller design and therefore simpler models that represent the non-linear are required.
- This provides the motivation for considering the use of NN in adaptive control.

Comment 5: Discussion on BP algorithm

Advantages

- the BP algorithm can be simulated on a PC so that no special hardware is required for its implementation.
- It is well known that pattern recognition is recently advancing towards control applications.
- versatile mapping capabilities from input to output.
- easy to apply using existing software packages e.g., Matlab neural network toolbox. Note: it is not as straightforward for control application as some have suggested.
- The researcher has the potential to explore the further capability of handling real complex non-linear problems.

Disadvantages

- BP learning is heuristic is *ad hoc* in nature, therefore it is time consuming.
- slow convergence speed (standard BP).
- sensitivity to initial conditions.
- Instabilities occur if the learning rate is too large.
- trapping in local minima.
- difficult to understand what is going on internally.

CHAPTER 5: ADAPTIVE CONTROL

5.1 Introduction

An adaptive control system uses a control scheme that is capable of modifying its behaviour in response to changes in process dynamics. Adaptive controllers have been extensively used in several industries including chemical, aerospace, automotive and pulp and paper industry [Silva, 1999]. However, the main objective of this chapter is to describe several fundamental concepts and results within the framework of adaptive control so that the ideas and techniques mentioned here can be linked directly or indirectly to early chapters especially Chapter 4 and the following Chapters 6.

5.1.1 Adaptive control schemes

A real-world plant can be usually characterised by time-varying dynamical properties most of them as a result of plant non-stationary, non-linear and random disturbances, which affect the plant behaviour in the following way:

- the plant is called non-stationary if its dynamics changes in time, e.g., as a result of ageing effects after being in operation for long time;
- if the plant is non-linear (as *every* real plant is) then the dynamical properties of its linearised model are different in the vicinity of various steady-state points; in normal operating conditions the steady-state point changes;
- Stochastic models are used to represent the disturbances acting at the plant output because of the large number and different nature of the factors disturbing the normal plant operation.

It is clear that the control algorithm used in the above circumstances should either be adaptive or should exhibit some robustness properties with respect to poor plant models and changes in the plant dynamics.

Robustness properties can usually be ensured by the feedback structure of the control system. The feedback compensates for the deviation of the plant output signal value from its set point, no matter which factor has caused such deviation:

(a) disturbances affecting the plant,
(b) improper plant model structure or
(c) a change in the plant model parameters.

However, the two latter factors usually cannot be dealt with well enough by the control system feedback structure alone. Large differences in plant model structure and large changes in the plant dynamics may cause the natural robustness properties of the control system to be exhausted thus causing unacceptable degradation of the system performance.

The adaptive control algorithms can also be used in such complex and difficult environment conditions. The main object of this type of control algorithms is to ensure such automatic change of the controller structure and parameters so that they correspond to the current properties of the plant and its environment to improve better robustness of the control system. According to [Astrom and Wittenmark, 1989], the usual methods of changing the controller parameters are:

i. programmed changes of the controller parameters, also known as gain scheduling;
ii. Identification of a plant model.

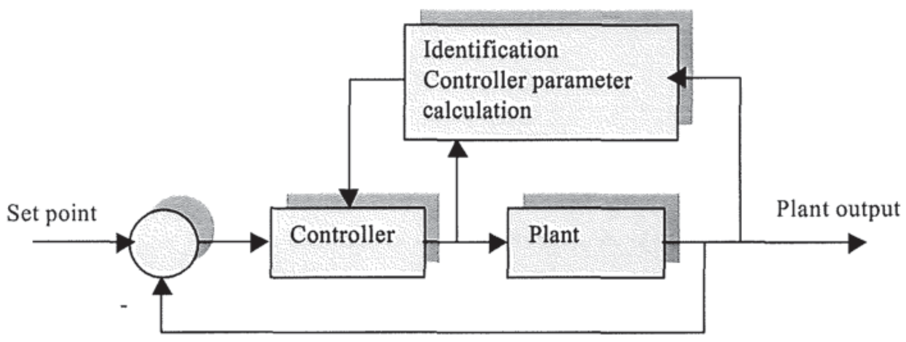


Fig.5.1 Block diagram of an adaptive control system with plant model identification

This chapter discusses the second type of adaptive control algorithm. The block diagram of such a control system is illustrated in Fig.5.1. The controller parameters are calculated as a result of recursive identification of the appropriate plant parametric model, performed on-line.

It is obvious that this control system structure has the potential to capture all changes in the plant parameters no matter what their origin. However, this scheme also has a rather important drawback:

- It is extremely complicated with respect to its theoretical analysis.

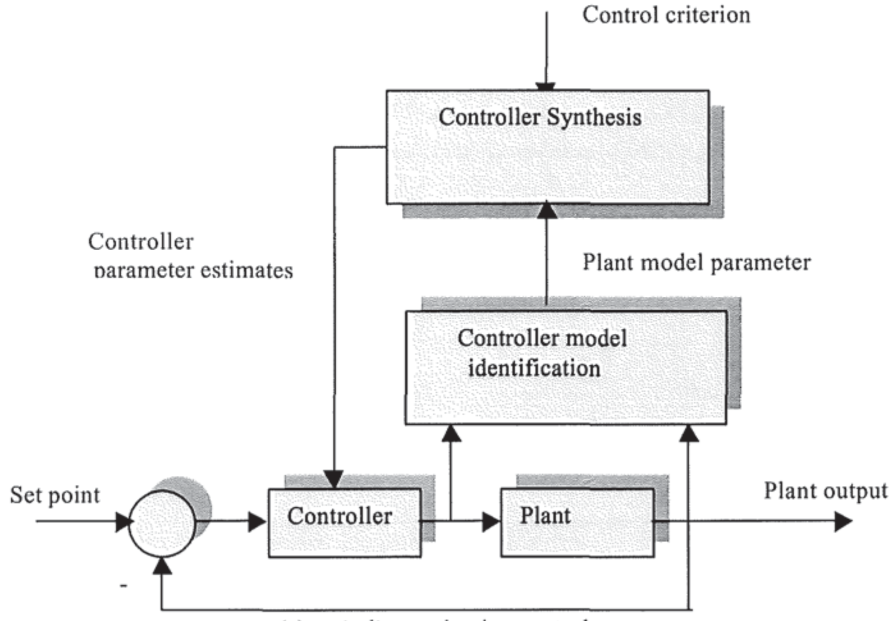
The complexity of the theoretical analysis results mainly due to non-stationary, non-linearity and stochastic disturbances. This is one of the primary reasons this research is carried out using MATLAB/SIMULINK and Tool boxes such as Neural Network, System Identification, Signal processing and Control toolboxes to investigate such adaptive systems.

Objective of this research is to investigate the following possibilities:

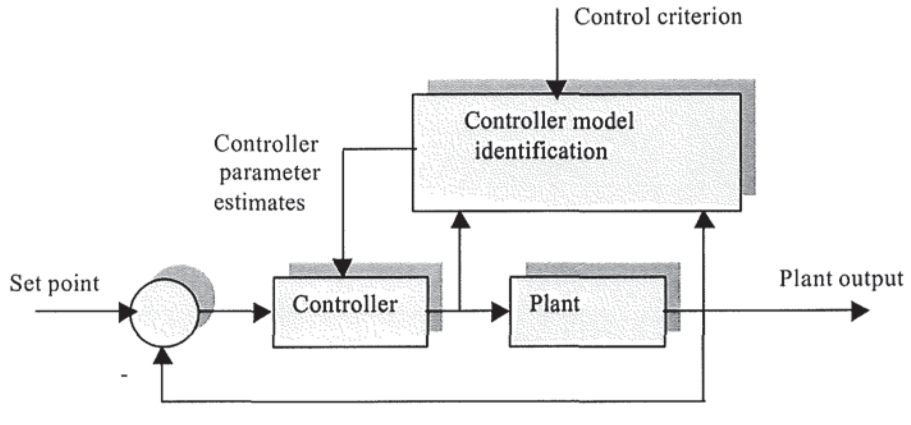
1. A model of the plant to be controlled as shown in Fig.5.2 (a). On the basis of the plant model and control criterion, we can proceed with control synthesis, i.e., calculate the parameters of the controller. This scheme is called *indirect adaptive control*, because to find the proper values of the controller parameters

we have to complete the intermediate model identification of plant.

2. It may be possible to identify the parameters of the controller that we are seeking. This scheme is called *direct adaptive control*, because it is possible to obtain directly the required controller parameters through their estimation in an appropriately redefined plant model. The block diagram of such an adaptive control system is presented in Fig.5.2 (b).



(a) an *indirect* adaptive control system



(b) *a direct* adaptive control system

Fig.5.2 Block diagram of an *indirect* and *direct* adaptive control system.

Both methods are known to have several important advantages as well as disadvantages. The main advantages of the *indirect* adaptive control scheme include:

- the general usefulness of the plant model obtained from identification;
- the model may be used for controller synthesis with several different control algorithms in mind;
- another useful property of this scheme is the possibility of starting identification while the plant is controlled by virtually any stabilising controller (for direct adaptive control the controller parameter identification is possible only if the currently tuned controller is working in the feedback channel of the system).

On the other hand, identification may be performed by simpler and more robust methods in the *direct* adaptive control system, as the controller synthesis has to be done only once and can be done *off-line*.

Assume that *direct* adaptive control is used for minimum-variance and pole/zero-placement controllers, while the *indirect* adaptive control scheme will be chosen for long-range predictive control algorithms.

5.1.2 Plant model

Adaptive control is usually used to cope with an *unknown* or *varying* plant to be controlled. Analysis and synthesis of such a control system is possible only under some assumptions concerning the nature of the plant and its dynamics. In this chapter only linear, discrete-time plants disturbed in a deterministic or stochastic manner is considered. The plant chosen is typical in the field of adaptive control and non-standard discrete-time control algorithms in general [Astrom & Wittenmark 1984; Astrom & Wittenmark 1989].

$$y(i) = z^{-k}\frac{B}{A}u(i) + \frac{C}{A\nabla^{l}}e(i) + d(i) + b \qquad (5.1)$$

where, the part $z^{-k}\frac{B}{A}u(i)$ referred to as the control channel of the plant. If

- $B = B^{+}$ (only stable factors exist in the B polynomial), the plant is called *minimum phase* (MP),
- $B = B^{+}B^{-}$ the plant is called *non-minimum phase* (NMP).

The part $\frac{C}{A\nabla^{l}}e(i) + d(i) + b$ is called the disturbance channel of the plant, with $\frac{C}{A\nabla^{l}}e(i)$ being stochastic part of the disturbance *d(i)* the deterministic part and *b* a constant bias of the plant output (deterministic in nature); the three parts represent all disturbances affecting the plant output.

In the discrete-time models of the plants, NMP zeros tend to be very common, mainly because:

- NMP zeros to the left of the unit circle in the *z*-plane are generated as a result of:
  - lack of synchronisation of the plant output sampling and changes in the control signal;
  - the presence in the continuous-time model of the plant of a time delay which is not an integer multiple of the sampling interval;
  - choice of too small a sampling interval;
  - choice of too large a discrete-model time delay for model parameters estimation;

- assumption of too large a number of poles compared with the number of zeros in the continuous-time plant model.

- NMP zeros to the right of the unit circle in the z-plane are generated as a result of the presence of NMP zeros in the continuous-time model of the plant, i.e. the plant itself is NMP. (not only its discrete-time model).

Stochastic disturbances are modelled as the output of a stable, invertible linear filter with the t.f. $\frac{C}{A\nabla^l}e(i)$. The filter is assumed to be excited with white noise *e(i)* of variance λ^2. Such disturbance is stationary for $l = 0$. For $l \geq 1$ the disturbance is non-stationary, i.e. it exhibits, for example, a changing mean value or ramp with changing slope, with its l^{th} difference being still stationary.

The following two kinds of C polynomials in the disturbance channel should be treated differently:

1. C polynomials passing the strict *positive realness* test, i.e.

$$\operatorname{Re}\left\{\frac{1}{C}-\frac{1}{2}\right\}\Bigg|_{z=e^{jwT_s}} > 0 \qquad \text{for all} \qquad 0 \leq wT_S \leq \pi \tag{5.2}$$

2. C polynomials violating the strict *positive realness* condition, i.e.

$$\operatorname{Re}\left\{\frac{1}{C}-\frac{1}{2}\right\}\Bigg|_{z=e^{jwT_s}} \leq 0 \qquad \text{for all some frequencies } 0 \leq wT_S \leq \pi \tag{5.3}$$

The *positive realness* condition is often essential while proving convergence of estimation schemes in the adaptive control system, e.g. recursive least squares in direct adaptive control or extended least squares method in indirect adaptive control.

It is generally agreed that the dynamical model, representing the deterministic disturbance as the response of a linear filter excited with the Kronecker delta function, assuming zero initial conditions, is the most suitable one for plant modelling:

$$d(i) = \frac{d}{A_g}\delta(i) \tag{5.4}$$

where, *d(i)* deterministic disturbance at the output of the plant; *d* disturbance amplitude; $A = A_g(z^{-1})$ deterministic disturbance generating polynomial;

$$\delta(i) = \begin{cases} 1 & for \quad i = 0 \\ 0 & for \quad i \neq 0 \end{cases} \qquad \text{Kronecker delta function.}$$

The zeros of the generating polynomial A_g determine the basic properties of the signal:

- zeros on the unit circle in the z-plane imply a non-vanishing disturbance signal;

- zeros inside the unit circle generate a vanishing deterministic disturbance;
- zeros outside the unit circle produce a bursting disturbance signal.

The constant bias *b* of the plant output is a special kind of deterministic disturbance; we may alternatively generate it using the polynomial $A_g = 1 - z^{-1}$. As the simplest kind of deterministic disturbance it was specially modelled in (5.1) and some special procedures for damping disturbances of this kind are also used in adaptive control system synthesis.

5.2 CONTROL ALGORITHMS SUITABLE FOR ADAPTIVE SYSTEMS

In this section some of the most popular adaptive control algorithms will be presented. The spectrum of an adaptive control algorithm described here does not pretend to be complete, rather some interesting adaptive control schemes also used in practice have been chosen.

Most control algorithms presented here could be described as difference equation of the form:

$$R(z^{-1})u(i) + S(z^{-1})y(i) - T(z^{-1})w(i) + h = 0 \tag{5.5}$$

where $u(i)$, $y(i)$ plant control signal and plant output signal respectively, $w(i)$ is set point of the plat output signal and *h* is the constant term.

The coefficients of the $R(z^{-1})$, $S(z^{-1})$ and $T(z^{-1})$ polynomials and the *h* term are chosen by the user before the simulation experiment and stay constant during the experiment.

5.2.1 Minimum-variance control algorithms

The aim of the minimum-variance control algorithm is the minimisation of the following performance [Astrom & Wittenmark, 1984; Astrom & Wittenmark 1989]:

$$J = E\left\{\left[P(z^{-1})\frac{\widetilde{B}^{-}(z^{-1})}{B^{-}(z^{-1})}y(i-K) - V(z^{-1})w(i)\right]^2 + q\left[Q(z^{-1})u(i)\right]^2\right\} \tag{5.6}$$

where $P(z^{-1})$, $Q(z^{-1})$ and $V(z^{-1})$ are filter polynomials affecting the minimized performance index signal components, *q* is a weighting coefficient for the control signal value $u(i)$, $\widetilde{B}^{-}(z^{-1})$ is reciprocal polynomial of the $B^{-}(z^{-1})$ polynomial, *K* is the prediction horizon in the performance index.

Minimisation of this performance index leads to a control algorithm of the same structure as (5.5). All parameters in the controller equation are identifiable from the prediction model (assuming $K=k$):

$$\begin{aligned}&\phi(i) - RB^{-}u(i-k) - SB^{-}y(i-k) + TB^{-}w(i-k) - hB^{-}(1)\\&=\{1-C\}[\phi(i) - Fe(i)] + Fe(i)\end{aligned} \tag{5.7}$$

where

$$\phi(i) = P\tilde{B}^{-}y(i) + \frac{q}{b_0^{+}}QB^{-}u(i-k) - VB^{-}w(i-k) \tag{5.8}$$

$$dR = \max(dB + K - 1, dQ + dC)$$
$$dS = \max(dA - 1, dP + dC - k)$$
$$dT = dV + dC$$

The generalized minimum-variance control algorithm presented above ensures that the resulting control system is stable for the *non-minimum phase* plant, even without having to use a non-zero q weighting coefficient and the $Q(z^{-1})$ polynomial, filtering the control signal in the minimised performance index (control signal value weighting is the standard way of stabilising the minimum-variance control system with NMP plant, but far from optimum).

Choice of minimum-variance control parameters

The following parameters of the minimum-variance control algorithm specification may be adjusted to guarantee the desired properties of the control system:

- the weighting coefficient for the control variable of the plant, which allows "soft" saturation of the control signal value and makes it possible to use this type of central algorithm for some NMP plants;
- the filter polynomial for the plant control signal, which allows the same kind of control signal saturation as described above, thus making it possible to use the minimum-variance control algorithm for some NMP systems; it also enables integral action to be introduced into the control system;
- the filter polynomials for the plant output signal and its set point in the performance index, which allow the introduction of integral action into the control system loop and the possibility of specifying a chosen model of set point followed by the plant output signal (modified tracking properties).

5.2.2 Pole/ Zero placement algorithms

As an example of pole/zero-placement control algorithms, some basic results for pole/zero-placement control of a NMP plant will be presented. The control algorithm is synthesized under the assumption that the control aim is to achieve the following transfer function (relating set point changes to plant output changes, cf.e.g. [Astrom & Wittenmark 1984; Astrom & Wittenmark 1989; Niederlinski & Moscinski, 1988].

$$K_m(z^{-1}) = z^{-k_m} \frac{K_m B^-(z^{-1}) B_m(z^{-1})}{A_m(z^{-1})} \tag{5.9}$$

where k_m is the discrete time delay in the desired model of set point following, $A_m(z^{-1})$ and $B_m(z^{-1})$ are the polynomials determining the desired poles and zeros of the $K_m(z^{-1})$ transfer function and the scalar coefficient K_m is: $K_m = \frac{A_m(1) B_m(1)}{B^-(1)}$

The effect of random disturbances on the plant output may be diminished by using a special form of observer polynomial $A_0(z^{-1})$, in the pole/zero placement control synthesis:

$$A_0(z^{-1}) = C(z^{-1}) A_0'(z^{-1})$$

where, $A_0'(z^{-1})$ is the basic observer polynomial, used mainly for adjusting the robustness properties of the resulting control system.

The parameters of the synthesized controller equation may be identified from the prediction model (assuming $k_m = k$):

$$\phi(i) - B^- Ru(i-k) - B^- Sy(i-k) + B^- Tw(i-k) - B^-(1)h = [1 - C][\phi(i) - R'e(i)] + R'e(i) \tag{5.10}$$

where

$$\phi(i) = A_0' A_m y(i) - A_0' B_m K_m B^- w(i-k) \tag{5.11}$$

$$dR' = k - 1 + dB^-$$
$$dR = k - 1 + db$$
$$dS = \max(dA - 1, dA_m + A_0 - k - dB^-)$$
$$dT = dB_m + dA_0$$

Choice of pole/zero placement control parameters

The following parameters of the pole/zero placement control algorithm may be adjusted to achieve the desired properties of the resulting control system:

- the polynomials defining the *poles* of the control system transfer function, determining the basic properties of the transient signals after set point changes and disturbance changes;
- the polynomials defining the *zeros* of the control system transfer function, modifying the tracking properties of the plant under control (only the stable part of the original plant $B(z^{-1})$ related dynamics could be modified for NMP plants).

5.2.3 Simple self-tuning control algorithms

A simple self-tuning control algorithm has been proposed by [Astrom, 1979]. This is essentially one of the pole-placement control algorithms, derived under the assumption that the plant under control may be described by the following simple model:

$$y(i) = -a_1 y(i-1) - a_2 y(i-2) + b_0 u(i-1) + b_1 u(i-2) + d \tag{5.12}$$

It is also assumed that the poles of the transfer function relating the changes in the set point signal with the changes in the plant output signal should be determined with the following formula:

$$z = e^{\xi w_0 T_p}\left[\cos\left(w_0 T_p \sqrt{1-\xi^2}\right) \pm j\sin\left(w_0 T_p \sqrt{1-\xi^2}\right)\right]$$

thus corresponding to the poles of the following Laplace transform transfer function:

$$s = -\xi w_0 \pm j w_0 \sqrt{1-\xi^2}$$

where ξ is the plant-damping ratio, and w_0 is the natural oscillation frequency of the plant. The parameters of this control algorithm may be easily identified from the properly configured prediction model.

This kind of adaptive control algorithm has become quite popular because of its similarity to the well-known, traditional design procedure for PI/PID controllers based on a simplified model of the plant (first or second order) and links to the family of pole-placement control algorithms. These two parameters of the simple self-tuning control algorithm as presented here for the demonstration purposes, while the relative damping and natural oscillation frequency can easily be interpreted in terms of the plant step-response properties.

Choice of the simple self-tuning controller parameters

The following parameters of the simple self-tuning control algorithm may be changed in order to influence the basic set-point-following properties of the resulting control system:

- control system damping ratio - allows determination of the damping ratio affecting the amplitude of the transient signals in the control system, after a set point change or in the presence of disturbances;
- transient signal natural oscillation frequency - allows determination of the frequency of oscillations in the control system transient signals, after a change of the set point value or in the presence of disturbances (the oscillation period should be measured in sampling intervals).

5.2.4 Multi-step Model Algorithmic Control (MAC)

Some basic predictive indirect adaptive control algorithms is described, including the *multi-step model algorithmic control* (MAC) and the *generalized predictive control* algorithm (GPC) in the next section.

The MAC control algorithm derivation is based on the assumption that the plant under control may be described with a weighting function approximate model and that the plant is undisturbed, i.e. *e(i =0*. The basic control system performance assumption, taken into consideration while deriving the MAC algorithm, is that the response of the plant output signal to a set point change should be the same as the response of the pre-specified first-order system of the form:

$$y^z(i) = z^{-k} \frac{1-\beta}{1-\beta z^{-1}} w(i) \tag{5.13}$$

where $y^z(i)$ is the desired trajectory of the plant response to the $w(i)$ set point changes and β determines the speed with which the change is followed.

The control algorithm aims at minimising the following performance index (corresponding to the above control system performance formulation):

$$J = \sum_{j=0}^{H} \left[\hat{y}(i+k+j|i) - y^z(i+k+j)\right]^2 + u^T Q u$$

where, $\hat{y}(i+k+j\|i)$	predicted value of the plant output at $(i+k+j)^{th}$ instant predicted at the i^{th} instant;
H	plant output prediction horizon;
$u = [u(i)u(i+1)\cdots u(i+H)]$	*V*ector of plant control signal values;
Q	plant control signal-weighting matrix.

By solving the stated minimization problem, the following control algorithm is obtained;

$$u(i) = -[1 \quad 0 \quad \cdots \quad 0]\left(G^T G + Q\right)^{-1} G^T r \tag{5.15}$$

where

$$G = \begin{bmatrix} g_0 & & & 0 \\ g_1 & g_0 & & \\ \vdots & \vdots & \ddots & \\ gH & gH-1 & \cdots & g_0 \end{bmatrix} \tag{5.16}$$

$$r^T = \begin{bmatrix} r_0 & r_1 & \cdots & rH \end{bmatrix} \tag{5.17}$$

$$r_s = \sum_{j=s+1}^{dB} \hat{g}_j u(i-j+s) - \beta^{s+1} y^M(i+k-1) - (1-\beta^{s+1})[y^M(i) - y(i) + w(i)] \tag{5.18}$$

with $g(i)$ being the *ith* element of the discrete-time pulse response of the plant and $y^m(i)$ the plant model output (in the form of a finite approximation of the plant weighting function):

$$y^M(i) = \sum_{j=0}^{dB} \hat{g}_j u(i-k-j) \tag{5.19}$$

Assuming additionally that, beginning from the $(i+k+L)^{th}$ time instant, the value of the control signal should be 0 (with L being the control signal predication horizon [Cutler, 1980], the following version of the control algorithm is obtained:

$$u(i) = -[1 \quad 0 \quad \cdots \quad 0](G_L^T G_L + Q)^{-1} G_L^T r \tag{5.20}$$

where

$$G_L = \begin{bmatrix} g_0 & & & 0 \\ g_1 & g_0 & & \\ \vdots & \vdots & \ddots & \\ g_L & g_{L-1} & \cdots & g_0 \\ \vdots & \vdots & & \vdots \\ g_H & g_{H-1} & & g_{H-L} \end{bmatrix} \tag{5.21}$$

5.2.5 GPC algorithm

During GPC algorithm derivation we assume that the plant dynamics may be described with the standard ARIMAX model, with the disturbance channel polynomial $C(z^{-1})$ being equal to 1 and $D(z^{-1}) = \nabla = 1 - z^{-1}$.

The objective of GPC algorithm is to minimize the performance index:

$$J = e^T e + q_0 \nabla u^T \nabla u \tag{5.22}$$

where e is the vector of control error defined as:

$$e^T = [w(i+k) - \hat{y}(i+k|i), \ldots, w(i+k+H) - \hat{y}(i+k+H|i)] \tag{5.23}$$

with $\hat{y}(i+k|i)$ the optimal predicted value of the plant output at the $(i+j)^{\text{th}}$ time instant predicted from the i^{th}, H the plant output prediction horizon. q_0 is the weighting coefficient of the vector of plant control signal values in the minimized performance index.

Minimization of performance index (5.22) leads to the long-range predictive control algorithm:

$$\nabla u(i) = [1 \quad 0 \quad \cdots \quad 0](H^T H + q_0 I)^{-1} H^T (w - r) \tag{5.24}$$

where

$$w^T = [w(i+k) \ldots w(i+k+H)] \tag{5.25}$$

is the vector of set point values and

$$\hat{y} = H\nabla u + r \tag{5.26}$$

$$H = \begin{bmatrix} h_0^k & & & 0 \\ h_1^{k+1} & h_0^{k+1} & & \\ \vdots & \vdots & \ddots & \\ h_H^{k+H} & h_{H-1}^{k+H} & \cdots & h_0^{k+H} \end{bmatrix} \tag{5.27}$$

$$\nabla u^T = [\nabla u(i) \ldots \nabla u(i+H)] \tag{5.28}$$

$$r^T = [r_0 \quad r_1 \quad \cdots \quad r_H] \tag{5.29}$$

$$r_s = \sum_{j=0}^{dA} g_j^{k+s} y(i-j) + \sum_{j=1}^{dB+K-1} h_{s+j}^{k+s} [u(i-j) - u(i-j-1)] \tag{5.30}$$

If we assume that starting from time instant $i+k+L$ the plant control signal u should be equal to 0, with L known as the control signal prediction horizon; we obtain the following simplified version of the GPC algorithm:

$$\nabla u(i) = [1 \quad 0 \quad \cdots \quad 0](H_L^T H_L + q_0 I)^{-1} H_L^T (w - r) \tag{5.31}$$

where

$$H_L = \begin{bmatrix} h_0^k & & & 0 \\ h_1^{k+1} & h_0^{k+1} & & \\ \vdots & \vdots & \ddots & \\ h_L^{k+L} & h_L^{k+L} & \cdots & h_0^{k+L} \\ \vdots & \vdots & & \vdots \\ h_H^{k+H} & h_{H-1}^{k+H} & \cdots & h_{H-L}^{k+H} \end{bmatrix} \tag{5.32}$$

5.2.6 Adaptive PID control with non-parametric identification

There are several methods for deriving the parameters of a PID controller on the basis of the controlled plant frequency response parameters. These methods include

- the Ziegler-Nichols frequency response method,
- the dominant poles design and
- the M_{max} method.

All of them are based on the determination of some points of the controlled plant frequency response, the plant critical gain and oscillation frequency being the most popular and easiest parameters to obtain. One of the most popular methods of approximate identification of the plant frequency response is based on the concept of closing the control loop in the tuning phase with a non-linear element of relay type, preferably with some hysteresis. During the initial phase of control, the relay in the feedback channel ensures that after some time oscillations will appear in the control system. The oscillations are continuously monitored and when their period seems to be constant we can measure it along with the oscillation amplitude. Afterwards the corresponding fundamental component is calculated. On the basis of such a modified Ziegler-Nichols experiment, the user knows all the data necessary to calculate the values of the PID controller parameters using the Ziegler-Nichols formulae. If we use a relay with hysteresis or any other more involved non-linear element in the feedback channel, we can gather more information about the plant frequency response parameters and obviously we can use more involved design procedures.

5.3 Identification in Adaptive Systems

In an adaptive control system, identification procedures are used to obtain the estimates of the parameters of the weighting function, transfer function or prediction model of the plant. The estimates of the prediction model parameters are either directly used in the adaptive controller equation or e.g., in the case of a NMP plant, the estimates are further processed to eliminate the influence of the NMP part of the plant model, $B^{-}(z^{-1})$.

The following three estimation schemes are commonly used in adaptive control systems:

(i) recursive least squares (RLS), (ii) recursive least mean squares (LMS), (iii) recursive prediction error minimization (RPE).

The basic and fundamental RLS and LMS procedures are briefly described below. But the extensive analysis and implementation techniques are given in [Ljung and Soderstrom, 1983].

5.3.1 Recursive least squares

The RLS method could be crudely implemented on the basis of the recursive equations:

$$\hat{\theta}(i) = \hat{\theta}(i-1) + k(i)\left[y(i) - \varphi^{T}(i-K)\hat{\theta}(i-1)\right] \tag{5.33}$$

$$k(i) = \frac{P(i-1)\varphi(i-K)}{\alpha + \varphi^T(i-K)P(i-1)\varphi(i-K)} \qquad (5.34)$$

$$P(i) = \frac{1}{\alpha}\left[P(i-1) - \frac{P(i-1)\varphi(i-K)\varphi^T(i-K)P(i-1)}{\alpha + \varphi^T(i-K)P(i-1)\varphi(i-K)}\right] \qquad (5.35)$$

where

$\hat{\theta}(i)$	model parameter estimates vector, in the *i*th identification step,
$\psi(i)$	model output variable value, in the *i*th identification step,
$\varphi(i)$	model regressor vector, in the i^{th} identification step,
K	prediction horizon (time delay) of the model,
$k(i)$	identification gain vector, in the i^{th} identification step,
$P(i)$	identification matrix (proportional to covariance of $\hat{\theta}(i)$, in the i^{th} identification step,
α	forgetting factor.

Initial values $\hat{\theta}(0)$ of the estimates are usually chosen as being equal to *0*. The initial identification matrix *P(0)* is usually a diagonal matrix with large coefficients on its main diagonal:

$$P(0) = P_0 I \qquad P_0 >> 1 \qquad (5.36)$$

If the forgetting factor in the RLS method is equal to 1, then the elements of the identification matrix *P(i)* decrease with the identification iterations, preventing rapid changes in the estimates after some iterations of the identification process. The possibility of changes in the estimates may be measured by the scalar gain coefficient

$$k_w = \varphi^T(i-K)P(i)\varphi(i-K) \qquad (5.37)$$

Rapid changes of the model parameter estimates are possible if either of the following modifications is used:

(a) covariance (***P*** matrix) resetting, periodically increasing the values of all *P(i)* matrix elements by such a factor that the largest element after the increase is equal to $\boldsymbol{P_o}$ (or $\frac{P_0}{\gamma}$), where γ is some scaling factor); such a mechanism makes it possible to increase the absolute values of the ***P*** matrix elements and the value of the scalar gain coefficient k_w, thus allowing fast (and effective) adaptation in the case of time-varying plant behaviour;

(b) using a forgetting factor $\alpha < 1$, causing the ***P(i)*** matrix elements calculated from (5.35) not to decrease much during the identification process.

Using a forgetting factor $\alpha < 1$ (if the controlled plant is not excited enough) causes the $\varphi(i-K)$ vector elements to be small and leads to constant increase of the *P(i)* matrix elements and the gain vector *k(i)*. As a result, when a large identification error happen (e.g. because of a sudden, large disturbance) the model parameter estimates $\hat{\theta}(i)$ change quite rapidly and disturb the process of identification. The effect may be especially disastrous in the direct adaptive control system, in which the estimates are directly used as) controller equation coefficients. Methods are described in the literature to avoid such unpleasant identification/control properties. For example, in [Albert and Sittler, 1966; Fortescue *et al.*, 1981; Kershenbaum and Fortescue, 1981] an estimator information measure is proposed, calculated as the weighted sum of the identification errors:

$$\Sigma(i) = \alpha(i)\Sigma(i-1) + [1-\varphi^T(i-K)k(i)][\psi(i) - \varphi^T(i-K)\hat{\theta}(i-1)]^2 \qquad (5.38)$$

The forgetting factor $\alpha(i)$ is adjusted at each identification step to keep the estimator information measure (5.38) constant, i.e. the user wants

$$\Sigma(i) = \Sigma(i-1) = \ldots = \Sigma(0) \qquad (5.39)$$

where $\Sigma(0)$ (the initial value of the $\Sigma(i)$) may be used for tuning the forgetting factor adjustment, in the presence of time varying plant behaviour.

Under the above assumptions, the forgetting factor for the ith step of identification is calculated by

$$\alpha(i) = 1 - [1-\varphi^T(i-K)k(i)][\psi(i) - \varphi^T(i-K)\hat{\theta}(i-1)]^2 \big/ \Sigma(0) \qquad (5.40)$$

Another method used to ensure good properties of the RLS estimation procedure for lengthy experiments is to keep the trace of *P(i)* constant and to introduce an identification error dead zone: if the identification error remains small enough, the update procedures are not activated at all. An adequate sequence of recursive formulae for this version of the RLS method is

$$\hat{\theta}(i) = \hat{\theta}(i-1) + \alpha(i)k(i)\left[\psi(i) - \varphi^T(i-K)\hat{\theta}(i-1)\right] \qquad (5.41)$$

$$k(i) = \frac{P(i-1)\varphi(i-K)}{1 + \varphi^T(i-K)P(i-1)\varphi(i-K) + \bar{c}\,\varphi^T(i-K)\varphi} \qquad (5.42)$$

$$\overline{P}(i) = P(i-1) - \alpha(i)\frac{P(i-1)\varphi(i-K)\varphi^T(i-K)P(i-1)}{1+\varphi^T(i-K)P(i-1)\varphi(i-K)+\overline{c}\varphi^T(i-K)\varphi} \tag{5.43}$$

$$P(i) = C_1\frac{\overline{P}(i)}{tr(\overline{P}(i))} + C_2 I \tag{5.44}$$

$$\alpha(i) = \begin{cases} \overline{\alpha} & for \qquad \left|\psi^T(i-K)\hat{\theta}(i-1)\right| > 2\delta \\ 0 & if \quad the-above-condition-does-not-hold \end{cases} \tag{5.55}$$

where

C_1 -	specified value of P matrix trace;	C_2 -	P matrix diagonalization factor,
δ -	Identification error dead zone;	$\overline{\alpha}$ -	Identification error weighting coefficient,
$\overline{c}$ -	a factor used to maintain sufficiently large values of the P matrix elements.		

A so-called improved least squares estimation method is also used. The convergence, stability and speed properties of this version are achieved by normalization of the regressor vector, scaling of the *P(i)* matrix to ensure the minimum condition number, adjusting the forgetting factor and avoiding updating the *P(i)* matrix and $\hat{\theta}(i)$ vector during periods of low excitation. The recursive calculations are as follows:

1. Normalization of the regressor vector:

$$n(i) = \max(1, \|\varphi(i-K)\|) \tag{5.45}$$

$$\psi_n(i) = \psi(i)/n(i) \tag{5.46}$$

$$\varphi_n(i-K) = \varphi(i-K)/n(i) \tag{5.47}$$

2. Calculating the forgetting factor value to stabilize *P(i)* matrix trace:

$$r(i) = 1+\varphi_n^T(i-K)P(i-1)\varphi_n(i-K) \tag{5.48}$$

$$\alpha(i) = 1-0.5\left[r(i)-\left\{r(i)^2-4\frac{\|P(i-1)\varphi_n(i-K)\|^2}{trP(i-1)}\right\}^{1/2}\right] \tag{5.49}$$

3. Testing the excitation properties

$$N(i) = \|P(i-1)\varphi_n(i-K)\| \tag{5.50}$$

If $N(i) < N_{min}$ (low excitation), the calculation proceed to 7.

4. Calculation of new value of *P*.

$$\overline{P}(i) = \frac{1}{\alpha(i)}\left[P(i-1) - \frac{P(i-1)\varphi_n^T(i-K)P(i-1)}{\alpha(i)+\varphi_n^T(i-K)P(i-1)\varphi_n(i-K)}\right] \tag{5.51}$$

$$\overline{P}^{-1}(i) = \alpha(i)P^{-1}(i-1) + \varphi_n(i-K)\varphi_n^T(i-K) \tag{5.52}$$

$$c\{\overline{P}\} = \|\overline{P}\|_\infty \|\overline{P}^{-1}\|_\infty \tag{5.53}$$

If $c\{\overline{P}\} > C_{\max}$ (poorly conditioned *P*), then the calculation proceeds to 6.

5. Updating the identified parameter vector and P.

$$P(i)=\overline{P}(i),\ P^{-1}(i)=\overline{P}^{-1}(i), S(i)=S(i-1) \tag{5.54}$$

$$\hat{\theta}(i)=\hat{\theta}(i-1)+S^{-1}(i-1)P(i)\varphi_n(i-K)*[\psi_n(i)-\varphi_n^T(i-K)\hat{\theta}(i-1)] \tag{5.55}$$

The calculation proceeds to 9.

6. Calculating the elements of the scaling matrices. $\overline{S}$ is calculated to minimize the condition number $\overline{SP}(i)$, then $\overline{P}(i)$, and $\overline{P}^{-1}(i)$, are updated:

$$\overline{P}(i)=\overline{SP}(i)\overline{S}, \qquad \overline{P}^{-1}(i)=\overline{S}^{-1}\overline{P}^{-1}(i)\overline{S}^{-1} \tag{5.56}$$

If $C\{\overline{P}\}<=C_{\max}$ (sufficiently well conditioned P), the calculation proceeds to 8.

7. Stopping the estimation

$$P(i)=P(i-1), \qquad \overline{P}^{-1}(i)=P^{-1}(i-1),\ \ S(i)=S(i-1) \tag{5.57}$$

$$\hat{\theta}(i)=\hat{\theta}(i-1) \tag{5.58}$$

8. Updating the identified parameters values and P:

$$P(i)=\overline{P}(i), \qquad P^{-1}(i)=\overline{P}^{-1}(i), \tag{5.59}$$

$$S(i)=S(i-1)\overline{S}, \qquad \varphi_n(i-K)=\overline{S}^{-1}\varphi_n(i-K), \tag{5.60}$$

$$\hat{\theta}(i)=\hat{\theta}(i-1)+S^{-1}(i)P(i)\varphi_n(i-K)[\psi_n(i)-\varphi_n^T(i-K)\hat{\theta}(i-1)] \tag{5.61}$$

9. Constraining the parameter estimates to some chosen neighbourhood of the initial guesses $\hat{\theta}(0)$. If R denotes the radius of a sphere in $\hat{\theta}$ space, in which all parameter estimates have to lie, then the estimates may be constrained:

$$\hat{\theta}_D(i)=\hat{\theta}(i)-\hat{\theta}(0) \tag{5.62}$$

$$\hat{\theta}(i)=\hat{\theta}(0)+\min\left(1,\frac{R}{\|\theta_D(i)\|}\right)\hat{\theta}_D(i) \tag{5.63}$$

10. In the improved least squares method, the user may choose the minimum value of the forgetting factor calculated by (5.49), the minimum value $N_{\min}$ of the norm in (5.50), the maximum value C_{max} of the condition factor and the maximum radius R in (5.63).

5.3.2 Recursive least mean squares

The recursive least mean squares (LMS) method is much simpler and faster than recursive least squares. The LMS method may be described by

$$\hat{\theta}(i)=\hat{\theta}(i-1)+\varphi(i-K)\frac{\gamma k}{\alpha k+\varphi^T(i-K)\varphi(i-K)}\left[\psi(i)-\varphi^T(i-K)\hat{\theta}(i-1)\right] \tag{5.64}$$

where

$\hat{\theta}(i)$	Parameter estimates vector,	$\psi(i)$	Model output,	$\varphi(i)$	regressor vector,
K	prediction horizon (time delay),	γk	gain coefficient,	αk	forgetting factor.

5.4 Identification model structure

In *indirect* adaptive control we must identify the parameters of a "classic" model of the plant, for example the weighting function or transfer-function model. The choice of model structure and identification scheme has been examined in details by many authors, see e.g. [Ljung, 1983].

Direct adaptive control needs a plant model in which the *R, S* and *T* polynomials and the *h* term are present, constituting the controller equation. This is the case for a *prediction model* of the plant, which is also linear in the parameters, making it possible to identify them using least squares.

The way in which the prediction model parameters are identified depends on the minimum phase property of the plant or on how information about the plant being non-minimum phase is incorporated.

5.4.1 Minimum phase plant

If the plant is minimum phase, the only unknown parameters in the prediction model are the resulting control algorithm parameters (with respect to which the model is linear). This means that the user may identify the parameters and use them directly as the controller coefficients. This way of implementing the minimum-variance and pole / zero-placement control algorithms is conceptually easy and numerically effective. Such a self-tuning control system is valid only for a minimum phase plant.

5.4.2 Non-minimum phase plant

If the plant is non-minimum phase, some elements in the corresponding prediction model will be related to this (specifically the B^- polynomial). An example (similar to the one described by (5.7)) is:

$$\phi(i) - RB^- u(i-k) - SB^- y(i-k) + TB^- w(i-k) - hB^-(1) = \in(i) \qquad (5.65)$$

where $\in(i)$ is the model error.

The estimation of such model-coefficient values should be effective and result in the controller parameters (*R, S* and *T* polynomials and the *h* term). This prediction model structure suggests at least two ways to achieve the stated objective:

- Identify jointly the parameters RB^-, SB^-, TB^- and $hB^-(l)$ in a prediction model

$$\phi(i) = Lu(i-k) - My(i-k) + Nw(i-k) - hB^{-} = \in (i) \qquad (5.66)$$

then conclude the controller parameters, *R, S, T* and *h*, by dividing *L, M* and *N* by B^{-} and $hB^{-}(I)$

- At the beginning of each identification step, filter the regression signals *y(i-k) and w(i-k)* with B^{-} and the constant 1 with $B^{-}(I)$, leading to the prediction model:

$$\phi(i) - Ru_{B^-}(i-k) - Sy_{B^-}(i-k) + Tw_{B^-}(i-k) - h1_{B^-} = \in (i) \qquad (5.67)$$

Such a model can be directly used to identify the control parameters: *R, S, T* and *h*.

These solutions assume that $B^{-}(z^{-1})$ is known. This may be so if either

- the user pretends to know B^{-}'s degree and coefficients on the basis of his/her plant knowledge (maybe as a result of previous identification); or

- B^- is identified jointly with the parameters of the controller; the possible solution is to use the prediction model:

$$\phi(i) - Lu(i-k) - Sy_{B^-}(i-k) + Tw_{B^-}(i-k) - h1_{B^-} = \in (i) \qquad (5.68)$$

At the end of each identification step, the unstable parts of *L* are grouped and processed further as B^-, the rest of the *L* estimate being treated as a controller polynomial *R*. In the next identification step, the estimate of B^-, serves to filter the regression variables *y(i - k)* and *w(i - k)* and the constant 1. In the first identification step we may assume that $B^{-}(z^{-1}) = 1$.

In specific cases some factors of the *R, S* and *T* polynomials may be known and thus need not be identified. If we denote such terms as R_k, S_k and T_k and with R_u, S_u and T_u, unknown, then during adaptive control the known factors of the controller polynomials should be used for filtering the appropriate regressor signals before the estimation step, while the unknown factors have to be identified (maybe jointly with the B polynomial).

5.5 Estimation of non-stationary plant parameters.

As is common practice in adaptive control and in CACSD-based adaptive control simulation, the user is free to choose among at least three estimation schemes:

(a) recursive least squares (RLS), in its standard version, the constant *tr P* version and improved version,

(b) recursive least mean squares (LMS),

(c) recursive prediction error minimisation (RPE).

The LMS method is considerably faster than both RLS and RPE. However, for many adaptive control systems, the RLS method is considerably better for parameter-estimate convergence rate and bias.

The basic parameter of weighted RLS is the forgetting factor α. The user should choose an equal to 1 for a plant assumed to be stationary, or $0 << \alpha < 1$ for a plant assumed non-stationary. Any constant a may cause slow convergence of the parameter estimates or periodic large changes of their values. Usually the forgetting factor may be continuously varied according to the weighted sum of squares of the model prediction error [Fortescue *et al.*, 1981]. The user may choose the lower bound of the adjusted forgetting factor and the gain of the adjustment procedure. Similar good properties may be obtained when using the constant *tr P* version of RLS [Astrom and Wittenmark, 1989] or the improved (or robust) least squares method [Sripada, 1987]. These methods ensure fast adaptation after a change of plant parameters and there is no danger related to low excitation of the plant (no "bursts" in the estimates).

The initial values of the main diagonal elements of *P* are also important in the RLS method. Initial values that are too large may cause temporary destabilization of the adaptive control system. On the other hand, small initial values may cause slow convergence of the parameter estimates.

RLS is especially vigorous in the first few iteration steps. This suggests restarting periodically with the initial P_o and the last vector of the parameter estimates (covariance resetting). Covariance resetting may be also requested if the user considers that the dynamics of the system have changed. The covariance resetting procedure makes RLS much faster in tracking changing parameters than the standard weighted RLS method with forgetting factor less than 1. On the other hand, covariance resetting may cause violent changes in the parameter estimates. It should be stressed that both the constant *tr P* version of RLS identification and improved least squares need no periodical resetting of the covariance, while achieving fast adaptation in the presence of non-stationary plant behaviour.

5.6 Multivariable Minimum-Variance Adaptive Control

Multivariable versions of the minimum-variance control schemes have appeared in the literature for almost more than 2 decades now, starting from the early works of [Borisson and Koivo, 1979]. Further work and extensions have been presented by, among others, [Dugard and Scattolini *et al.*, 1985].

It is supposed that the multivariable plant to be controlled is described by the linear model:

$$A(z^{-1})y(i) = B(z^{-1})u(i-k) + C(z^{-1})e(i) + d \qquad (5.69)$$

with $u(i)$ a p-dimensional control signal, $y(i)$ a p-dimensional output (controlled) signal, $e(i)$ a p-dimensional white noise sequence with diagonal covariance matrix, and d a p-dimensional constant disturbance (bias) vector. $A(z^{-1})$, $B(z^{-1})$ and $C(z^{-1})$ are polynomial matrices, with A being monic. Argument (z^{-1}) will be omitted from here on.

It is assumed that the plant is minimum phase in the multivariable sense, i.e. that B is stable in such sense. It is also assumed that C is stable.

The performance index to be minimized in the multivariable generalized minimum-variance scheme is

$$\min_{u(i)}\left[E\left\{\left\|y(i+k)-P^{-1}Vw(i)\right\|^2+\left\|Qu(i)\right\|^2\right\}\right] \tag{5.70}$$

where $w(i)$ denotes a p-dimensional set point vector and P, V and Q are p x p matrices in the z^{-1} operator.

The control algorithm resulting from this problem statement is defined by

$$\left[\overline{F}B+\overline{C}B_0^{-T}Q_0^TQ\right]u(i)+\overline{G}y(i)-\overline{C}w^*(i)+\overline{F}(1)d=0 \tag{5.71}$$

$$C=AF+z^{-k}G \tag{5.72}$$

$$\overline{C}F=\overline{F}C \tag{5.73}$$

$$\overline{F}G=\overline{G}F \tag{5.74}$$

$$Pw^*(i)=Vw(i) \tag{5.75}$$

with F and $\overline{F}$ p xp polynomial matrices of k-1 degree and G and $\overline{G}$ also p x p polynomial matrices, of degree sufficient for the identity (5.72) to be solvable.

It is apparent that the control equation may be rewritten in the following, commonly adopted form:

$$Ru(i)+Sy(i)-Tw^*(i)+\delta=0 \tag{5.76}$$

Thus, the control vector could be calculated using

$$u_{opt}(i)=-\overline{R}_0{}^{-1}\left[\sum_{j=1}^{nR}\overline{R}_0u_{opt}(i-j)+\sum_{j=0}^{nS}\overline{S}_jy(i-j)-w^*(i)\sum_{j=1}^{nT}\overline{T}_jw^*(i-j)+\delta\right] \tag{5.77}$$

with

$$R=\overline{F}B+\overline{C}B_0^{-T}Q_0^TQ \tag{5.58}$$

$$S=\overline{G} \tag{5.59}$$

$$T=\overline{C} \tag{5.60}$$

$$\delta = \overline{F}(1)d \tag{5.61}$$

It should be observed that, in the case of no control signal weighting (i.e. $Q = 0$), the assumption concerning the minimum phase property of the controlled plant becomes active.

It should be observed also that the plant model might be transformed to include the polynomial matrices appearing in the multivariable minimum-variance control algorithm just given. The appropriate version of the plant model is

$$\begin{aligned} &\psi(i+k) - \left[\overline{F}B + \overline{C}B_0^{-T} Q_0^T Q\right]u(i) - \overline{G}y(i) + \overline{C}w^*(i) - q \\ &= (\overline{C} - 1)[Fe(i+k) - \psi(i+k)] + Fe(i+k) \end{aligned} \tag{5.82}$$

with

$$\psi(i+k) = y(i+k) - w^*(i) + B_0^{-T} Q_0^T Q u(i) \tag{5.83}$$

The model (5.82) could then be rewritten in the prediction model form (regressor model form):

$$\psi(i) - R_u R_k u(i-k) - S_u S_k y(i-k) + T_u = y(i+k) - w^*(i) + B_0^{-T} Q_0^T Q u(i) \tag{5.84}$$

or, assuming that some factors of the controller polynomial matrices are known, in the form:

$$\psi(i) - R_u R_k u(i-k) - S_u S_k y(i-k) + T_u = y(i+k) - w^*(i-k) - q = \in(i) \tag{5.85}$$

in which only the unknown factors and vectors R_u, S_u, T_u and q need be estimated, if this prediction model is used in the multivariable adaptive control framework. The controller polynomial matrices appearing in (5.85) are of special structure and different dimensions compared with R, S and T, e.g. R_u, and R_k are defined as

$$R_u = \begin{bmatrix} R_{u,11} & & 0 & \cdots & R_{u,1P} & & 0 \\ & \ddots & & & & \ddots & \\ 0 & & R_{u,P1} & \cdots & 0 & & R_{u,PP} \end{bmatrix} \tag{5.86}$$

$$R_u = \begin{bmatrix} R_{k,11} & & 0 \\ & \ddots & \\ R_{k,p1} & & 0 \\ \vdots & \ddots & \\ 0 & & R_{k,1p} \\ & \ddots & \\ 0 & & R_{k,pp} \end{bmatrix} \tag{5.87}$$

Prediction model (5.85) together with the controller equation (5.77) form the basis of the multivariable adaptive minimum-variance control scheme.

5.7 Multivariable Predictive Adaptive Control

Below, a multivariable case of generalised predictive control will be considered as a representative of multivariable versions of long-range predictive control algorithms.

Multivariable versions of the GPC control algorithm emerged as natural generalizations of the basic GPC control algorithm, their background being traceable to [Dugard *et al.,* 1985; Goodwin *et al.,* 1980]. The version presented here resembles that presented by [Shah, *et al.,* 1987] as well as similar results obtained by [Kinnaert, 1987; Gu *et al.,* 1991].

It is assumed that the plant to be controlled is described by the multivariable model:

$$A\Delta y(i) = B\Delta u(i-k) + e(i) \tag{5.88}$$

where *d* denotes the diagonal z^{-1} operator matrix with all diagonal elements difference operators $1 - z^{-1}$, all other elements of the model having been introduced in the previous section (5.2.5).

The performance index to be minimized in step *i* of the GPC algorithm is

$$J_V(i) = E\left\{ \sum_{j=N_1}^{N_2} \left\| y(i+j) - w(i+j) \right\|^2 + \sum_{j=0}^{N_2} \left\| \rho\Delta u(i+j) \right\|^2 + \right\} \tag{5.89}$$

where ρ is the control weighting matrix and $\Delta u(i+j)$ vector serves as a vector of variables with respect to which the performance index is minimized.

In practice, the following related performance index will be used in subsequent calculations:

$$J(i) = \sum_{j=N_1}^{N_2} \left\| \hat{y}(i+j|i) - \hat{w}(i+j|i) \right\|^2 + \sum_{j=0}^{N_2} \left\| \rho\Delta u(i+j) \right\|^2 \tag{5.90}$$

where $\hat{y}(i+j)$ denotes the vector of optimal predictions of the controlled variables, assuming at time instant *i* zero values of all future white noise sequence elements; $\hat{w}(i+j)$ denotes the vector of set point predictions. Two cases could be considered:

- Perfect knowledge of the future set point sequence is available, in which case $\hat{w}(i+j|i) = w(i+j)$.
- No knowledge of the future set point sequence is available, giving $\hat{w}(i+j|i) = w(i)$ as the only reasonable solution.

The following observations, concerning the assumptions just imposed, should be made:

1 Usually, in long-range predictive control, a receding horizon concept is adopted, i.e., the optimization performed at each instant i gives the whole vector of "optimal" control increments, only the first element of which is used in practice, the whole calculation procedure being repeated in each step, producing new, updated values of consecutive u signal increments.

2 Usually we assume that after some specified number of control periods, the increments of the control signal should be equal to zero, resulting in the upper index of the sum relating to u in (5.90) being N_u, instead of N_2:

$$J(i) = \sum_{j=N_1}^{N_2} \left\|\hat{y}(i+j|i) - \hat{w}(i+j|i)\right\|^2 + \sum_{j=0}^{N_u} \left\|\wedge \Delta u(i+j)\right\|^2 \tag{5.91}$$

This so-called control horizon may in practice be much less than the controlled variable prediction horizon (or simply prediction horizon) N_2, greatly simplifying and accelerating the calculations required.

3. All prediction horizons, N_1, N_2 and N_u were given as scalars above. However, nothing prevents them from being defined as appropriate p-dimensional vectors, corresponding to the assumed structure of the multivariable plant model. In what follows, the scalar version of the presentation relating to the prediction horizons will be retained, so as not to complicate the notation.

Let us define the following aggregate vectors:

$$\hat{y}^*(i) = [\hat{y}(i+N_1|i)^T\, \hat{y}(i+N_1|i)^T \ldots \hat{y}(i+N_2|i)^T\,]^T \tag{5.92}$$

$$\hat{w}^*(i) = [\hat{w}(i+N_1|i)^T\, \hat{w}(i+N_1|i)^T \ldots \hat{w}(i+N_2|i)^T\,]^T \tag{5.93}$$

$$\Delta u^*(i) = [\Delta u(i)^T\, \Delta u(i+1)^T \ldots \Delta u(i+N_u)^T\,]^T \tag{5.94}$$

The performance index to be minimized (5.91) may be rewritten using the new notation as

$$J(i) = \left\|\hat{y}^*(i) - \hat{w}^*(i)\right\|^2 + \left\|\rho\Delta u^*(i)\right\|^2 \tag{5.95}$$

The vector of predicted values of the controlled signal could be shown to be

$$\hat{y}^*(i) = H\Delta u^*(i) + r(i) \tag{5.96}$$

with $\Delta u^*(i)$ defined as in (5.94), i.e. including only future increments of the control vector, while the influence of all past values of the control vector on the vector of predicted values of the controlled signal is grouped in the vector $r(i)$ vector, of dimension $(N_2 - N_1)$ x p. Thus it is clear that the predicted values of the controlled signal $\hat{y}*(i)$, with the constraint that all the future increments of the control signal are zero ("free" prediction), would be *r(i)*, giving easy on-line calculation of the elements of vector *r(i)*, if the model of the multivariable plant is known.

The H matrix appearing in (5.96) is the plant model step response block matrix:

$$H = \begin{bmatrix} h_{N_1-1} & & & 0 \\ h_{N_1} & h_{N_1-1} & & \\ \vdots & \vdots & \ddots & \\ h_{N_u} & h_{N_u-1} & \cdots & h_0 \\ \vdots & \vdots & & \vdots \\ h_{N_2-1} & h_{N_2-2} & \cdots & h_{N_2-1-N_u} \end{bmatrix} \tag{5.97}$$

with each h_i element a p x p matrix of i^{th} elements of the multivariable plant step response. Incorporating (5.96) into the performance index (5.91) and completing the minimization results in the final control algorithm:

$$\Delta u^*_{opt}(i) = (H^T H+)\rho(i) - r(i)) \tag{5.98}$$

from which the required current control increments $\Delta u_{opt}(i)$ may be extracted.

We can conclude that, for adaptive multivariable GPC control, the following steps can be adopted in each iteration of the multivariable plant control:

1 Identify the parameters of the plant model in the form of (5.88).

2 Calculate the estimates of the multivariable plant step response H matrix, in the form defined by (5.97), on the basis of the plant model obtained in step 1.

3 Calculate the estimates of the multivariable plant "free" response prediction vector *r(i)*, as defined by (5.96), also on the basis of the plant model identification.

4 Calculate the optimal control increments using (5.98) and extract the first p elements of the $\Delta u^*(i)$ vector.

5.8 Multivariable System Identification

System identification methods are inevitably present in the adaptive control systems and usually affect their behaviour quite substantially. A lot of work in the adaptive control field in practice is devoted to making the underlying identification schemes faster, more reliable and more robust. It is agreed that the multivariable identification case can be even more difficult.

In what follows some concepts concerning multivariable identification methods will be outlined. However, the coverage does not pretend to be either very elaborate or very thorough. The methods given below follow closely the standard ones, which may be found for example in [Ljung, 1983], with some numerically oriented enhancements as suggested, for example, in [Lawson and Hanson, 1974].

Basically, we can use two different kinds of multivariable plant model:

- a *plant-oriented model*, also called a transfer-function model (following the SISO case), as described, for example, by (5.88); this kind of model is used in multivariable GPC;
- a *controller-oriented model*, also called a prediction model, introduced by (5.82), (5.84) and eventually (5.85); this kind of model is used for multivariable generalized minimum-variance control.

In both cases, the plant model to be identified is of the general form:

$$\psi(i) - \hat{\Theta}^T \Phi(i) = \in (i) \tag{5.99}$$

with $\psi(i)$ the output of the model, $\hat{\Theta}^T$ the model parameter matrix, $\Phi(i)$ the regressors vector, and $\in (i)$ the model error vector. For the plant-oriented model case, $\psi(i)$ is simply $\Delta y(i)$, as in (5.88), whereas $\Phi(i)$ is formed from the delayed $\Delta y(i)$ and $\Delta u(i)$ signal increment vectors, with A and B polynomial matrices building $\hat{\Theta}^T$.

Estimation is usually performed through consecutive rows of the (5.99) model, i.e. the p-sub-models of the form:

$$\psi^j (i) - \hat{\Theta}^T \Phi(i) = \in^j (i) \tag{5.100}$$

with $\psi^j (i)$ the j^{th} output of the model, $\hat{\Theta}^{j^T}$ the jth row of the parameter estimate matrix $\hat{\Theta}^T$ and $\in^j (i)$ the j^{th} element of the model error vector. Estimation then proceeds through the well-known formulae, e.g. in weighted recursive least squares, the calculations during each step are:

$$\hat{\Theta}^j (i) = \hat{\Theta}^j (i) + k(i)\left[\psi^j(i) - \hat{\Theta}^{jT} (i-1)\phi(i)\right] \tag{5.101}$$

$$k(i) = \frac{P(i-1)\phi(i)}{\alpha + \phi^T (i) P(i-1)\phi(i)} \tag{5.102}$$

$$P(i) = \frac{1}{\alpha}\left[P(i-1) - \frac{P(i-1)\phi(i)\phi^T (i) P(i-1)}{\alpha + \phi^T (i) P(i-1)\phi(i)}\right] \tag{5.103}$$

where *k(i)* is the identification gain vector, *P(i)* the identification (covariance) matrix and $\alpha$the forgetting factor.

To enhance the robustness of the estimation scheme as well as the speed of identification / adaptation, many improvements have been suggested in the literature. Considering the formulae (5.101)-(5.103), some of the most promising ones are for example, the constant trace version of the RLS estimation, [Astrom and Wittenmark, 1989], and the improved (robust) version of it introduced by [Sripada and Fisher, 1987].

One important feature should be observed in these formulae:

- the regressor vector $\phi(i)$, the gain vector $k(i)$ and the identification (covariance) matrix $P(i)$ are the same for all p identification procedures for the p rows of the $\hat{E}$ parameter matrix. That means that in each identification step, the updating of $P(i)$ and $k(i)$ has to be performed only once applying (5.102) and (5.103). This version of $P(i)$ and $k(i)$ updating is sometimes called common *regressors estimation*.

In the controller-oriented model, the $\psi(i)$ vector is $\psi(i)$ of the model (5.85), and the regressor vector can be formed from the delayed and filtered $u(i)$, $y(i)$ and $w^*(i)$, as well as from the vector of unity elements. The parameter matrix $\hat{E}$ can be formed from the polynomial matrices R_u, S_u and T_u as well as the vector q. In this case, identification should proceed as in the plant-oriented model case, i.e. using the common regressor estimation scheme.

However, it is obvious that the special structure of *unknown* and *known* factors of R, S and T polynomial matrices can be utilized, as may be observed in formulae (5.86) and (5.87). Indeed, the number of identified parameters can be substantially reduced by removing zero elements from the *unknown* and *known* factors and performing the structure compression. However, by doing so, we encounter a situation in which the $\phi^j(i)$ regressor vectors are different for parameter estimation of the different rows of $\hat{E}$. This specialized version of the multivariable RLS estimation scheme is called *independent regressor* estimation.

5.9 PERFORMANCE MEASURES FOR ADAPTIVE CONTROL ALGORITHMS

It seems reasonable to work out general and precise performance measures for adaptive control algorithms, which can be used to compare the results obtained by different adaptive schemes. Using such measures, we could answer the following typical questions:

Q1 Does the new algorithm offer better performance when compared with existing ones with respect to adaptive system?

Q2 Which values of existing adaptive control schemes should we choose in order to achieve a stable and optimal adaptive system?

It is obvious that each stochastic adaptive control algorithm should ensure successful (optimum) control of a linear time-invariant stochastic plant, if possible. The concept of *maximum regulability*, as defined below, can be used to gain performance-related information about the possibly adaptively controlled plant:

$$r_{\max} = \boxed{\frac{\{\text{Uncontrolled plant output signal variance}\} - \{\text{Controlled signal minimum variance}\}}{\{\text{Uncontrolled plant output signal variance}\}}}$$

Maximum regulability is a measure of the relative output variance decrease achievable in the best case, i.e. when a known plant (minimum phase or not) is controlled by the properly tuned minimum-variance controller. The maximum regulability indicates how much can be gained by controlling a stochastically disturbed, linear time-invariant plant. A small value of the maximum regulability index suggests that practically nothing can be gained by controlling that plant, because the small decrease of the output variance could well be below the noise variance of the measurement instruments.

The maximum regulability values for minimum phase (MP) plants can be computed using formulae first presented by [Astrom, 1966]:

$$r_{\max} = \frac{E\left\{\left[\frac{G}{A}e(i-k)\right]^2\right\}}{E\left\{\left[\frac{C}{A}e(i-k)\right]^2\right\}}$$

From the definition of r_{max} leads to draw the following properties:

- r_{max} for minimum phase (MP) plants depends only on the time delay k and the properties of the disturbance filter $\frac{C}{A}$;
- r_{max} for NMP plants depends additionally upon the B- polynomial, which could be justified on the basis of theoretical similarities between time delay and NMP;
- r_{max} for NMP plants is generally smaller than for their nearest MP neighbours, which could be concluded also from the similarities just mentioned;
- r_{max} for white noise at the plant output we get $\frac{C}{A}=1$, $G = 0$ and $r_{max} = 0$ which means a minimum of maximum regulability: the white-noise power spectrum is too broad for any control system to cope with;
- r_{max} is always smaller than 1, because in the best circumstances (i.e. for $k = 1$) the controlled variable is reduced to the driving white noise $e(i)$;
- r_{max} is invariant with respect to the white noise variance;
- with increasing time delay k, all other things being equal, r_{max} decreases to 0.

Another performance measure for adaptive control algorithms is the price paid for minimizing the output variance. This measure can be defined as the variance of the plant control signal for minimum-variance control and can be computed from

- for MP plants: $E\left\{u^2(i)\right\} = E\left\{\left[\frac{G}{B^+} e(i)\right]^2\right\}$
- for NMP plants: $E\left\{u^2(i)\right\} = E\left\{\left[\frac{G}{B^+ \overline{B}^-} e(i)\right]^2\right\}$

It is also reasonable to define the relative minimum-variance control cost c_{Mv} as:

$$c_{Mv} = \frac{\{\text{Control variance for minimum-variance controlled plant}\}}{\{\text{Uncontrolled plant output signal variance}\} - \{\text{Controlled signal minimum variance}\}}$$

thus relating the increase in the control signal variance to the corresponding decrease of the controlled signal variance.

All adaptive control algorithms are built on the basis of corresponding non-adaptive ones. It seems reasonable to define a performance measure with respect to perfectly calculated control algorithms of the chosen kind, i.e. assuming perfect knowledge of the plant structure and parameters as well as plant disturbance characteristics. Such a measure will be called the *obtainable regulability* r_o and is defined as

$$r_o = \frac{\{\text{Uncontrolled plant output signal variance}\} - \{\text{Controlled plant output signal variance}\}}{\{\text{Uncontrolled plant output signal variance}\}}$$

It is obvious that

$r_0 \le r_{\max}$

Similarly it is worthwhile to account for control variable changes by introducing the obtainable relative control cost:

$$c_o = \frac{\{\text{Control variance for the controlled plant}\}}{\{\text{Uncontrolled plant output signal variance}\} - \{\text{Controlled plant output signal variance}\}}$$

These performance measures defined above may be evaluated for any control algorithm supposed to be implemented in the adaptive control system for the given plant.

The last concept concerns the truly adaptive control algorithm and determines its closeness to its non-adaptive origin. The *adaptive regulability* (r_a) could thus be defined as:

$$r_a = \frac{\{\text{Uncontrolled plant output signal mean square}\} - \{\text{Adaptively controlled plant output signal mean square}\}}{\{\text{Uncontrolled plant output signal mean square}\}}$$

Adaptive regulability should be determined in the stationary phase of the adaptive control system simulation, i.e. after the controller has converged. It is obvious that:

$r_a \le r_o$

Proceeding analogously we can define the relative obtainable control cost as a performance reference value for adaptive control based on a given non-adaptive control law. The relative *adaptive control cost* c_a, could be defined as:

$$c_a = \frac{\{\text{mean square control signal for the a daptively controlled plant}\}}{\{\text{Uncontrolled plant output signal mean square}\} - \{\text{Adaptively controlled plant output signal mean square}\}}$$

The r_a and c_a indices referred respectively to r_o and c_o give together an insight into how much are we behind the non-adaptive control performance in terms of quality and cost.

5. 10 SIMULATIONS STUDIES

The main objective of simulation studies is to demonstrate the analysis and synthesis of minimum-variance and predictive adaptive control systems algorithms developed in this chapter using MATLAB and SIMULINK. No MATLAB toolboxes are directly used for this exercise. All examples provided are user written as MATLAB m-files. Controllers are elements of the simulated control systems are prepared as S-functions. MATLAB allows creation of sophisticated software even though elements of this software are "hidden" in the SIMULINK structure. The S-functions realize all identification and synthesis of controllers. Only a few function from the ControlToolbox and Signal Processing Toolbox are incorporated within S-functions (See appendix C).

Adaptive GPC- tracking properties

Adaptive generalised predictive control (GPC) serves as an example of indirect adaptive control (See Fig.5.2 (a) in section 5.1.1). Thus, the common difference from the previously discussed minimum variance control is the necessity of identification of the plant model and recalculation of the controller parameters. GPC provides integral action. However, the quality of the reference signal tracking depends heavily on the parameters of the GPC controller (See Fig.5.4).

The main purpose of the following examples is to show that different control and output prediction horizon can provide stable behaviour of the system. The examples show the ability to reject both deterministic and stochastic disturbances.

Example 1: Correct structure of the model:

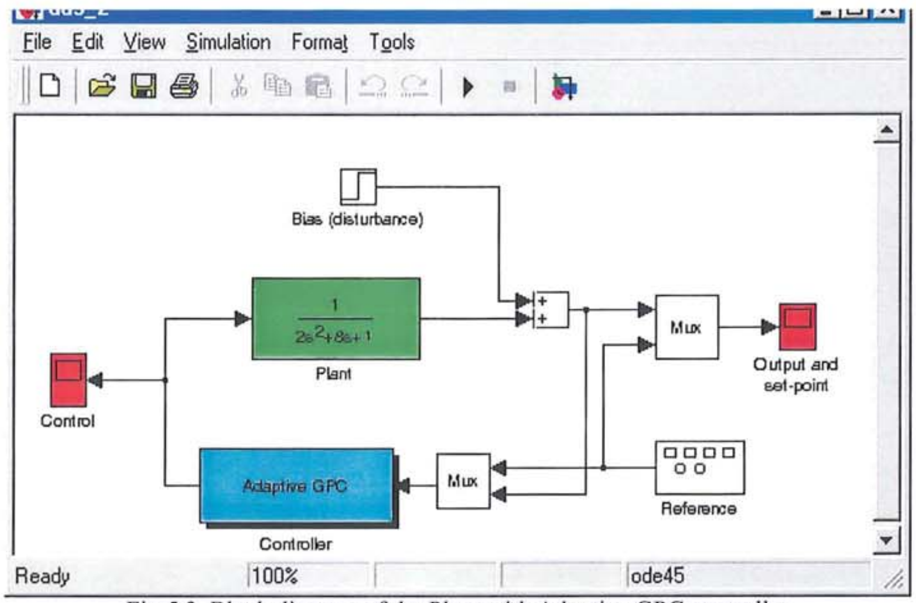
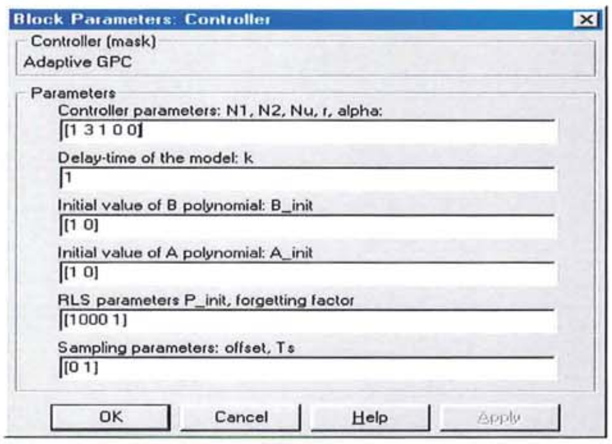


Fig.5.3 Block diagram of the Plant with Adaptive GPC controller



Block Parameters: Controller

Controller (mask)
Adaptive GPC

Parameters
Controller parameters: N1, N2, Nu, r, alpha:
[1 3 1 0 0]
Delay-time of the model: k
1
Initial value of B polynomial: B_init
[1 0]
Initial value of A polynomial: A_init
[1 0]
RLS parameters P_init, forgetting factor
[1000 1]
Sampling parameters: offset, Ts
[0 1]

Fig.5.4: Adapive GPC parameter block

A constant disturbance affecting the output of the plant to be controlled is one of most typical in practice. The disturbance is often called a *bias*. This example shows control quality when the plant is disturbed with a bias disturbance. Parameters of the controllers are set as: Fig.5.4: r = 0, alpha=0 (i.e., no control weighting and no feed-forward precompensation. Where control algorithm defines as: *N1*- minimal horizon of output prediction; *N2*- maximal horizon of output prediction; *Nu* - control prediction horizon; *r*-weight on control increments; *alpha* - tuning parameter for slowing down the reference trajectory; *k* - assumed delay time of the plant; *B_init* - initial value of polynomial B of linear model; *A_init* - initial value of polynomial A of linear model; *P_init* - initial value on diagonal of covariance matrix (RLS); *forgetting_factor* - forgetting factor of RLS ; *offset*, *Ts* - offset and sampling time respectively. In this simulation the algorithm parameters were chosen heuristically to provide the best performance with the second order plant.

The simulated performances of example 1, alongside its controller effort are plotted in Figure 5.5 and 5.6 respectively. This demonstrates that adaptive GPC is able to track the set-point reasonably well. The controller has converged to a stable state and is robust to the bias disturbance. This is however dependent on the selection of optimal parameters.

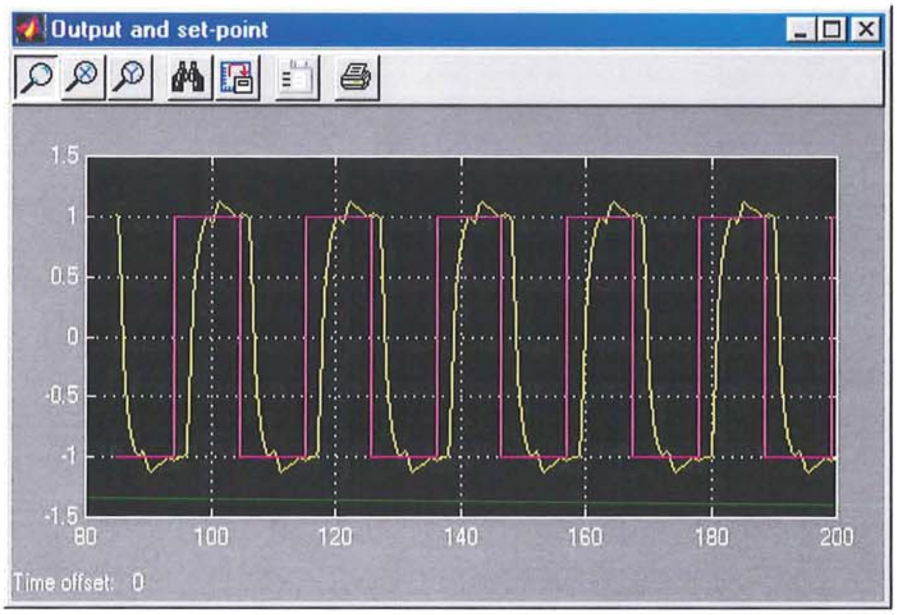


Fig.5.5 Performance plot of Output and Set point

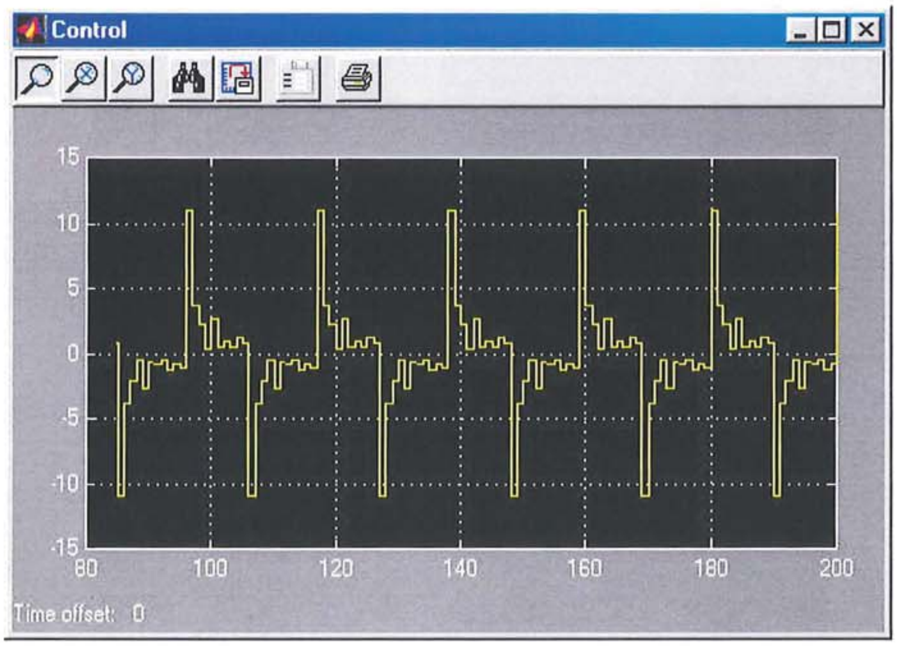


Fig.5.6 Control output

Example 2: Rejection of stochastic disturbance

The examples compare the quality of stochastic disturbance rejection in three cases:

1. no control
2. GPC control
3. Minimumvariance (MV) control.

It is recommended to wait until transient phase vanishes.

Generalization of minimum-variance control in the sense of the GPC algorithm causes the quality of stochastic disturbance rejection to deteriorate compared with MV. However, it is well known that proper performance of MV needs an almost perfect model, while predictive control is much more robust. Anyway, it is worth checking how predictive control can cope with stochastic disturbances. Program is used to check how the quality of the rejection of stochastic disturbances deteriorates when compared with minimum-variance control. The block diagram of the overall simulation is in figure 5.7(a) with component sub-functions and parameter values in figure 5.7(b-e).

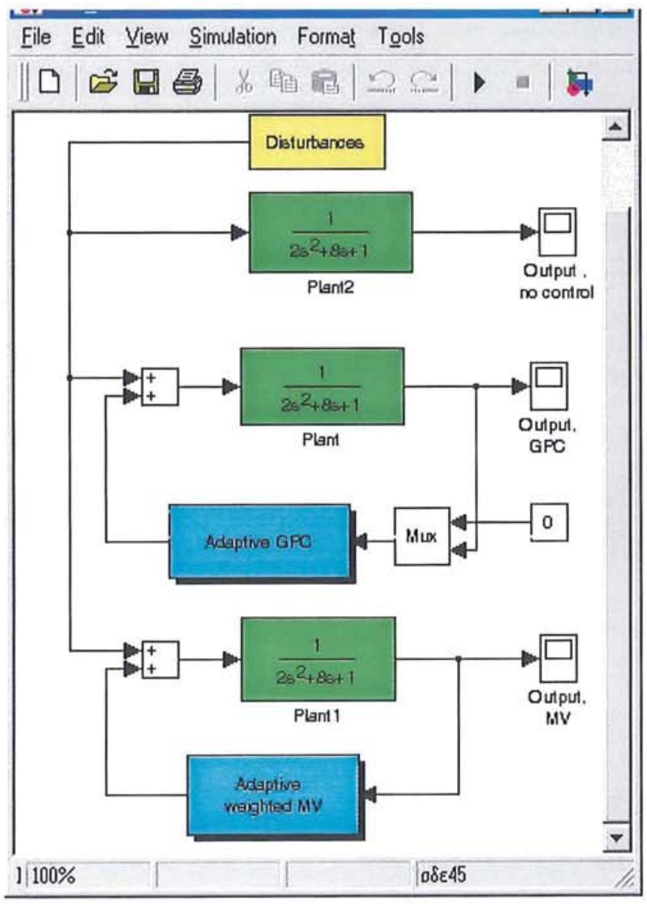


Fig. 5.7(a) Block diagram of plant with adaptive GPC and MV controllers.

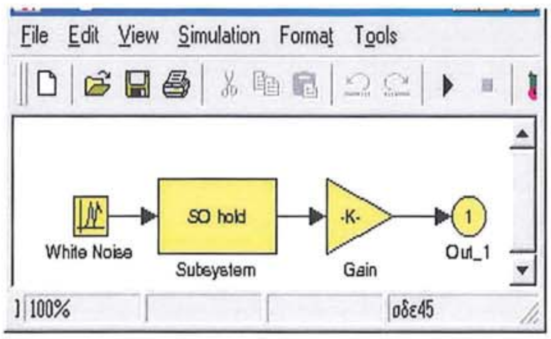


Fig. 5.7(b) Block diagram of noise sub-systems

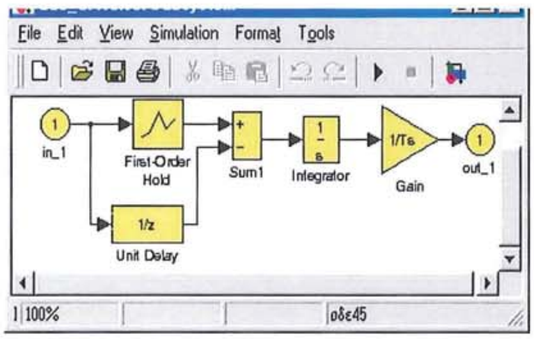
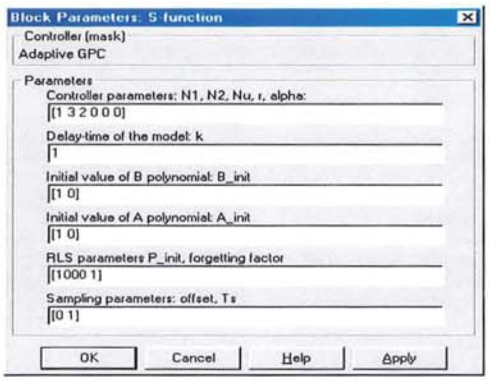


Fig. 5.7(c) Block diagram for noise sub-sub-system



Block Parameters: S-function

Controller (mask)

Adaptive GPC

Parameters

Controller parameters: N1, N2, Nu, r, alpha:

[1 3 2 0 0 0]

Delay-time of the model: k

1

Initial value of B polynomial: B_init

[1 0]

Initial value of A polynomial: A_init

[1 0]

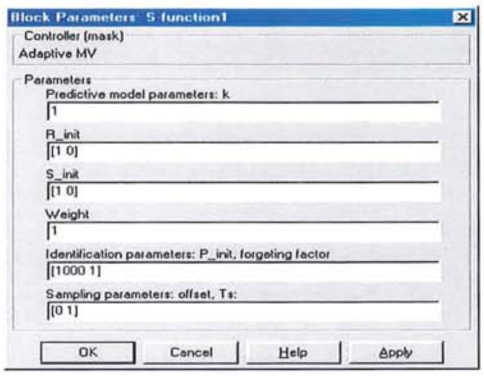
RLS parameters P_init, forgetting factor

[1000 1]

Sampling parameters: offset, Ts

[0 1]

Fig. 5.7(d) Adaptive GPC parameters



Block Parameters: S-function1

Controller (mask)

Adaptive MV

Parameters

Predictive model parameters: k

1

R_init

[1 0]

S_init

[1 0]

Weight

1

Identification parameters: P_init, forgeting factor

[1000 1]

Sampling parameters: offset, Ts:

[0 1]

Fig. 5.7(e) Adaptive MV parameters

Camparison BEFORE:

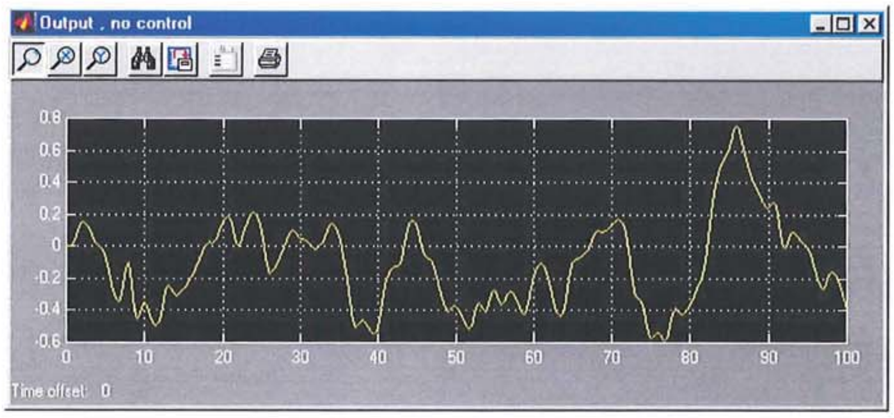


Fig 5.8(a) Plant response

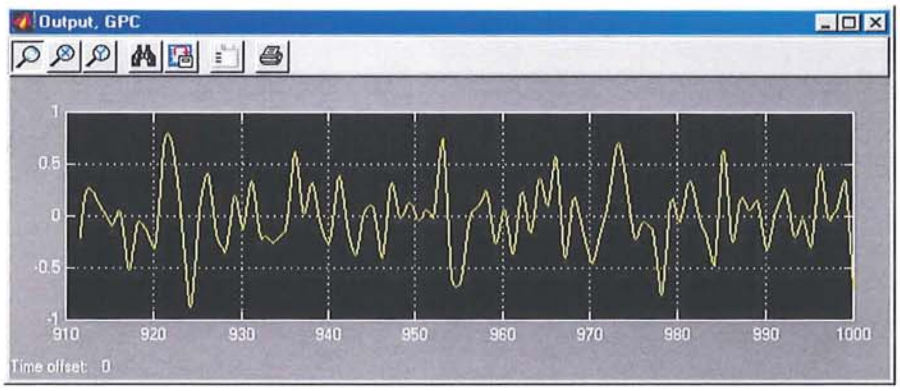


Fig 5.8(b) Adaptive GPC initial response

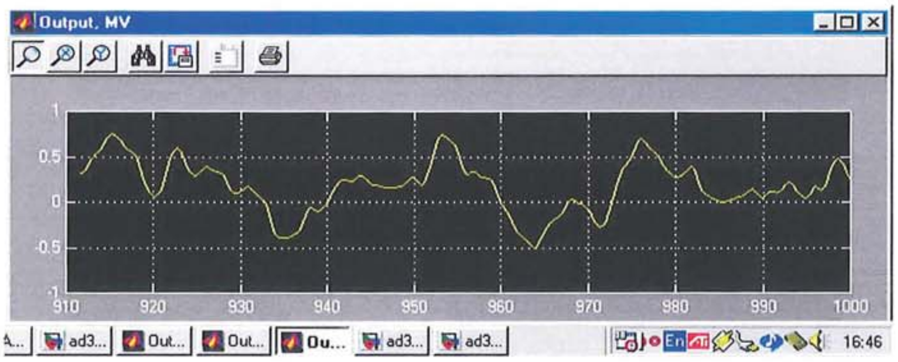


Fig 5.8(c) Adaptive MV initial response

Figure 5.8(a) shows the plant response to the stochastic disturbance of example 2. This is the target trajectory for both types of adaptive contoller to follow. In this inital stage of the simulation the adaptive MV controller output (figure 5.8(c))is less susceptible to the disturbance than that of the the adaptive GPC output of figure 5.8(b).

AFTER: Only changing following GPC parameters: r =1 (instead of 0), Ts = 2 (instead of 1) improves the response it indicating that GPC is more robust and flexible than MV. In this part of the experiment the adaptive GPC controller response of figure 5.9(c) tracks the plant output more closely than the adaptive MV controller does.

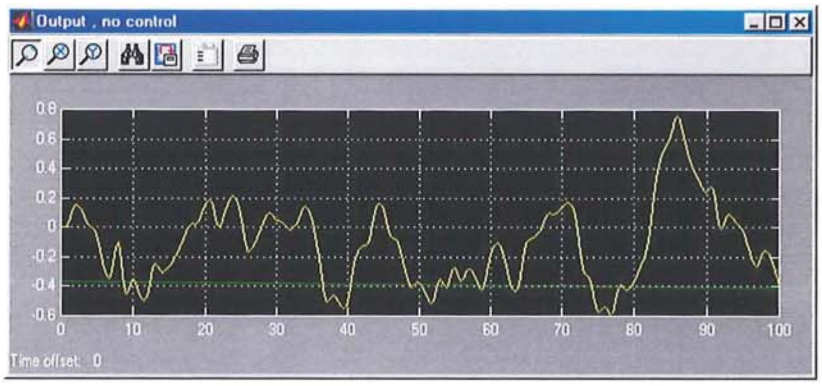


Fig 5.9(a) Plant response

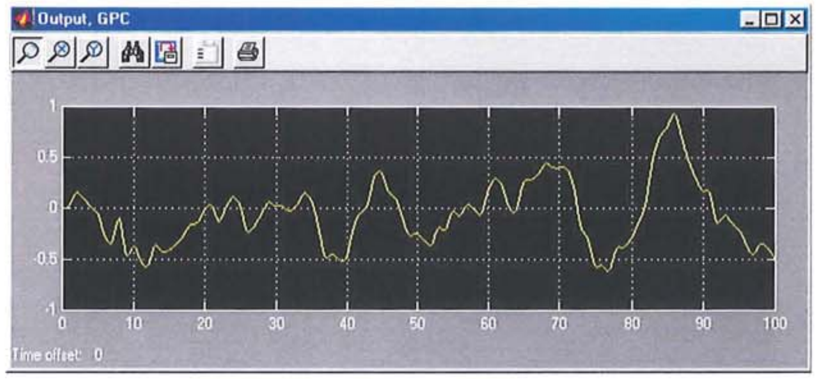


Fig. 5.9(b) Adaptive GPC subsequent response

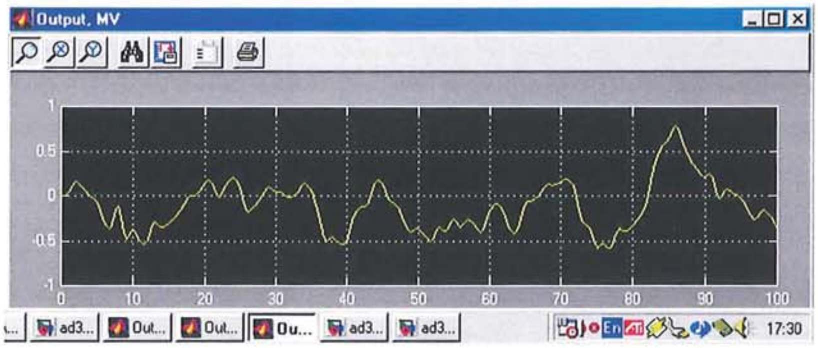


Fig 5.9(c) Adaptive MV susequent repsonse

5.11 CONCLUSIONS

In this chapter several concepts and results concerning adaptive control systems were delivered. The field of stochastic adaptive control has gained a lot of interest among control research and development groups during the past few years. Several different adaptive control algorithms have emerged, many of which are heuristic and almost all of which lacking proofs of control system stability and convergence of the parameter estimates. This is due to complication of the analysis and synthesis of adaptive control. Hence Matlab and SIMULINK is used for simulation.

CHAPTER 6: NEURAL NETWORK ENHANCED GENERALISED MINIMUM VARIANCE SELFTUNING CONTROL FOR NON-LINEAR DISCRETE-TIME SYSTEMS

6.1 Introduction

The main of objective of this chapter is to present a neural network enhanced self-tuning controller, which is designed by amalgamating neural network mapping with a generalised minimum variance self-tuning control (GMVSTC) strategy. Using this technique, the controller can deal with non-linear plant that exhibit a variety of features such as; uncertainties, non-minimum phase, coupling effects and unmodelled dynamics (assumed to be globally bounded). The unknown non-linear plants to be controlled are approximated by an equivalent model, which is composed of simple linear plus non-linear sub-models respectively.

The generalised recursive least squares algorithm is used for identifying the linear submodel and a layered neural network is applied to detect the unknown non-linear submodel where the weights are updated based on the error between the plant output and the output of the linear sub-model as shown in Fig.6.1.

Since the controller design method is based on the equivalent model, the non-linear sub-model is naturally accommodated within the control law.

The effectiveness of the control algorithm is illustrated by means of simulation examples.

6.2 Brief Overview of Backpropagation

Backpropagation (BP) was created by generalizing the Widrow-Hoff learning rule to multiple-layer networks and non-linear differentiable transfer function. Input vectors and the corresponding output vectors are used to train a network until it can approximate a function, associate input vectors with specific output vectors or classify input vectors in an appropriate way as defined by you. Networks with biases, a sigmoid layer and a linear output layer are capable of approximating any function with a finite number of discontinuities.

Standard BP is a gradient descent algorithm, as is the Widrow-Hoff learning rule (See Chapter 4 for detail). The term BP refers to the manner in which the gradient is computed for non-linear multilayer networks. There are a number of variations on the basic algorithm, which are based on other standard optimization techniques, such as conjugate gradient and Newton methods.

Properly trained BP networks tend to give reasonable answers when presented with inputs that they have never seen. Typically, a new input will lead to an output similar to the correct output for input vectors used in training that are similar to the new input being presented. This generalization property makes it possible to train a network on a representative set of input target pairs and get good results without training the network on all possible input/output pairs.

6.2 Adaptive control for non-linear system

Most studies on adaptive control for non-linear system [Taylor *et al.* 1996; Sastry and Isidori, 1989; Kopoulos *et al.* 1992] depend on a common assumption that the non-linear plant under consideration can be represented by:

$$y(t) = \sum_{i=1}^{n} \theta_i x_i(t) \tag{6.1}$$

where $y(t)$ is the plant output at discrete time steps $t \in 1,2,..n$, $x_i(t)$ are the known non-linear functions augmented with delayed plant output signals $y(t-1)...y(t-n_y)$ and delayed plant input signals $u(t-1)...u(t-n_u)$ and θ_i contains the parameters associated with $x_i(t)$. A model of (6.1), which is linear in the parameters can be estimated on line using a generalised recursive least squares algorithm and the new controller output $u(t)$ can be obtained by solving a set of non-linear equations [Zhu *et al.*, 1999].

MNNs is applied as a tool for modelling non-linear dynamic functions due to their ability to approximate complex output and input non-linear relationships sufficiently [Irwin *et al.* 1995]. Using a BP neural network learning algorithm [Rumalhart *et al.*, 1989], neural network have presented a popular architecture in many research fields, such as non-linear system identification and control, signal processing, pattern classification and so on [more details refer to Chapter 4]. Nielsen, 1989, presented analytical results showing the capability of layered neural networks to approximate non-linear functions, which under certain conditions, give any $\varepsilon > 0$, there exists a three layered neural network so that any function can be approximated to with ε mean squared error accuracy. More recent results shows that such a networks can exactly match input/output behaviour [Sontag, 1998].

Multilayer neural networks application studies to non-linear discrete-time adaptive control systems can be found in [Chen 1990; Narendra, 1990], and a convergence result for adaptive regulation using MNN is presented in [Chen and Khali, 1992], where many restrictive assumptions have been made. To cope with the problems encountered, a modified scheme with dead-zone was introduced by [Chen and Khalil, 1992]. However some of the following problems still remains unsolved [Zhu *et al.*, 1999]:

(a) It is difficult to generate overall controller convergence when the plant is in nonminimum phase. This is particularly awkward for discrete time control in which nonminimum phase behaviour can be brought about by the sampling rate selection.
(b) Only tracking problems can be successfully dealt with.
(c) Optimum performance cannot be assured, when a performance index is not inherently considered in the algorithm design.

(d) The learning rate for MNNs is relatively slow, leading to a failure to deal successfully with rapidly changing operating points.

In order to tackle these above problems, an enhanced self-tuning controller algorithm is introduced by [Zhu *et al.*, 1999] which combines a neural network identification method with a generalise minimum variance (g.m.v) control strategy. In this control algorithm, an unknown non-linear plant is represented by an equivalent model, which consists of a simple linear plus non-linear submodel. This type of model seems to be particularly useful in an adaptive control framework. Standard recursive algorithm technique is applied to identify the parameter of the linear sub-model and BP network is used to update the weights of the NNs, which is based on the error between the plant output and linear submodel output (see Fig. 6.1).

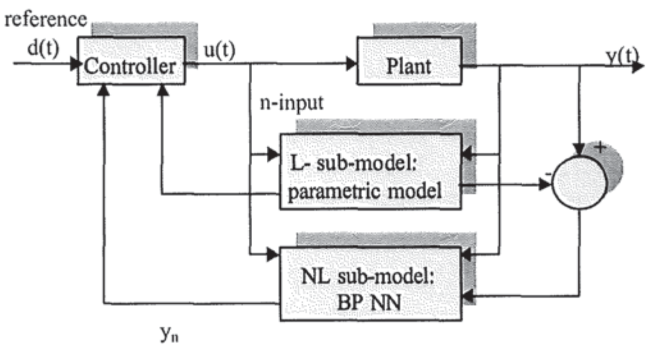


Fig.6.1: Block diagram of the control system structure

where: *d(.)* is the reference, *y(.)* is the plant output, *u(.)* is the controller output to the plant input $y_l(.)$ is the linear sub-model output, y_n is the neural network output.

6.4 Controller Design Methodology

The plant being investigated can be described by:

$$\text{Plant:} \quad y(t+1) = f(Y,U) \tag{6.2}$$

where $f(.) \to R^n$; $\{Y \in R^{n_y}; U \in R^{n_u}; n = n_y + n_u\}$ is a complicated smooth (i.e., infinitely differentiable) non-linear function and $y(t) \in Y$ and $u(t) \in U$ are the plant output and input signals, respectively, at discrete times $t \in 1,2,...n$. To control such a non-linear plant, a generalised parametric time varying plant model structure is used [Zhu *et al.* 1999].

$$\text{Model:} \quad A(z^{-1})y(t+1) = B(z^{-1})u(t) + f(Y,U) \tag{6.3}$$

where $A(z^{-1})$ and $B(z^{-1})$ are n_y and n_u order polynomials, z^{-1} is a one step backward shift operator. Also, it is considered that the parameters associated with polynomial $A(z^{-1})$ and $B(z^{-1})$ are either time invariant or are slowly time varying. Function $f(.) \rightarrow R^n$ is potentially a non-linear function, therefore the equivalent model is a combination of a linear time varying plus a non-linear submodel respectively.

In order to calculate a g.m.v.control form, the performance index can be described as [Zhu *et al.*, 1999]:

$$J = \left|e(t+1)\right|^2 = \left|S(z^{-1})y(t+1) - Rd(t) - Q(z^{-1})u(t) - Hf(.,.)\right|^2 \qquad (6.4)$$

where $d(t)$ is a bounded reference input. Further $S(z^{-1})Q(z^{-1})$, R and H all weighting polynomials of z^{-1}.

Now define an auxiliary (secondary) output $\phi(t+1) = S(z^{-1}) = C(z^{-1})y(t+1)$ where its optimal predictor is given by:

$$\phi^*(t+1/t) = G(z^{-1})y(t) + C(z^{-1})B(z^{-1})u(t) + C(z^{-1})f_0(.,.) \qquad (6.5)$$

in which $G(z^{-1})$ and $C(z^{-1})$ satisfy:

$$S(z^{-1}) = C(z^{-1})A(z^{-1}) + z^{-1}G(z^{-1}); \qquad n_g = n\text{-}1 \qquad (6.6)$$

where n_c and n_g are the orders of polynomials $C(z^{-1})$ and $G(z^{-1})$ respectively. Note that (6.5) can be obtained by multiplying both side of (6.3) with $C(z^{-1})$.

The solution, which minimises the performance index (6.4), is then found to be:

$$\phi^*(t+1/t) = Rd(t)y(t) + Q(z^{-1})u(t) + Hf_0(.,.) \qquad (6.7)$$

From (6.5) and (6.6), the controller output $u(t)$, which satisfies the minimisation requirement can be obtained from:

$$u(t) = \frac{Rd(t) + Hf_0(.,.) - G(z^{-1})y(t) - C(z^{-1})f_0(.,.)}{-Q(z^{-1}) + C(z^{-1})B(z^{-1})} \qquad (6.8)$$

By substituting the control-input (6.8) into (6.3) the closed loop system can be expressed as:

$$[B(z^{-1})S(z^{-1}) - Q(z^{-1}) + A(z^{-1})]y(t+1) = B[Rd(t) + Hf_0(.,.) - C(z^{-1})]f(.,.) \qquad (6.9)$$

To obtain desired closed dynamics, a most likely stable polynomial $T(z^{-1})$ is assigned in advance as:

$$T(z^{-1}) = t_0 + t_1 z^{-1} + ... + t_{n_t} z^{-n_t} \tag{6.10}$$

The following relationship therefore needs to be accommodated in order to achieve the required action.

$$\begin{aligned} &B(z^{-1})S(z^{-1}) - Q(z^{-1})A(z^{-1}) = T(z^{-1}); \\ &n_\xi = n_q = n - 1; n_t \leq 2n - 1 \end{aligned} ; \tag{6.11}$$

So $S(z^{-1})$ and $Q(z^{-1})$ are found from (6.11) in which, at the time of calculation, $A(z^{-1}), Q(z^{-1})$ and $T(z^{-1})$ are all known. To cancel the static offset, R may be chosen, most simply as [Zhu *et al.*, 1999]:

$$R = \frac{T(1)}{B(1)} \tag{6.12}$$

Finally, to eliminate the effect, in the steady state of the non-linear part, H can be chosen as:

$$H = \frac{Q(1)}{B(1)} \tag{6.13}$$

The block diagram of the control system structure is shown in Fig.6.1.

If parameters associated with the polynomials $A(z^{-1})$ and $B(z^{-1})$ and the non-linear function $f_0(.,.)$ are known, the above algorithm may be directly applied and the system output $y(t+k)$ will exactly track the reference input signal with satisfactory performance. However, in practice, the parameters and non-linear function are most likely unknown *a priori*. Therefore, a recursive algorithm can be used here on line to estimate the linear submodel parameters and a BP network is used to approximate the non-linear function. The neural network of the structure is as shown in Fig.6.2.

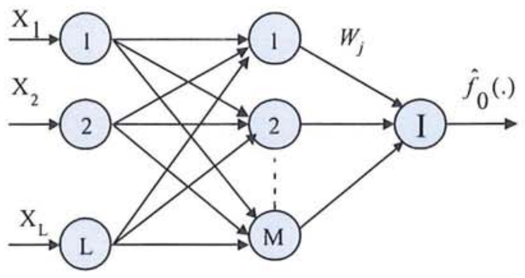


Fig.6.2: FFNN model to approximate non-linear function $f_0(.,.)$

The identification of the plant model of (6.3) consists of two parts, linear submodel parameter estimation and non-linear function approximation. For the linear submodel parameter estimation a recursive algorithm can be used based on:

$$\hat{y}(t+1) = \varphi(t)^T \hat{\theta}(t) + \hat{f}_0(.,.) \tag{6.14}$$

where $\varphi(t) \in R^{2n}$ is a data vector, $\hat{f}_0(.,.)$ is the model error and $\theta$is the parameter vector.

It is apparent that f_0 will later be revealed as a double agent in terms of its role as an approximation to the non-linear function f_0 [Zhu *et al.*, 1999].

Thus with:

$$\hat{\theta}(t) = [\hat{a}_1, \ldots \hat{a}_n; \hat{b}_0, \ldots \hat{b}_n] \tag{6.15}$$

and

$$\varphi(t) = [y(t), \ldots, y(t-n_y); u(t), \ldots, u(t-n_u)]; \tag{6.16}$$

the updated procedure is:

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \frac{P(t)\varphi(t)H_s(z^{-1})}{1+\varphi(t)^T P(t)\varphi(t)} v_0(t+1) \tag{6.17}$$

with

$$P(t+1) = \frac{P(t)}{\lambda_1(t)} - \frac{P(t)\varphi(t)\varphi(t)^T P(t)}{\lambda_1(t) + \lambda_2(t)\varphi(t)^T P(t)(t)^T P(t)} \tag{6.18}$$

Here $0 < \lambda_1(t) \le 1; 0 \le \lambda_2(t) \le 2$ and:

$$\begin{aligned} v_0(t+1) &= H_s(z^{-1})[y(t+1) - \varphi(t)^T \hat{\theta}(t) - \hat{f}_0(.,.)] \\ &= H_s(z^{-1})\{[\theta - \hat{\theta}(t)]^T \varphi(t) + [f_0(.,.) - \hat{f}_0(.,.)]\} \end{aligned} \tag{6.19}$$

in which $H_s(z^{-1})$ is a convergent filters, $H_s(z^{-1}) = H_{s1}(z^{-1}) / H_{s2}(z^{-1})$;
$H_{s1}(z^{-1})$ and $H_{s2}(z^{-1})$ are stable polynomials with $H_{s1}(0) = 1$ and $H_{s2}(0) = 1$.

A BP NN is used to approximate the non-linear model error $\hat{f}_0(.,.)$, which has three layers, the input layer neurones being n_i, the hidden layer n_h sigmoidal neurone. The network is therefore eminently suitable for non-linear function modelling [Warwick, 1995].

The task in tuning the network is to adjust all variable weights such that the error E_k can be reduced where the error is defined as:

$$E_k = \frac{1}{2}[\hat{f}_0(.,.) - f_0(.,.)]^2 \tag{6.20}$$

Alternatively equation (6.14), and the plant model (6.3) can be described as:

$$y(t+1) = \varphi(t)^T \theta + f_0(.,.) \tag{6.21}$$

which is an equivalent model with the same dynamics as the original. As part of the identification exercise in each sampling period, its linear parameters are estimated by the recursive algorithm. The non-linear function *f₀(.,.)* can therefore be numerically detected by:

$$f_0(.,.) = y(t+1) - \varphi(t)^T \hat{\theta}(t)]^T \tag{6.22}$$

The error training signal of the neural network may be provided from (6.22), whilst the input signal of the network is given by:

$$I = [y(t-1),\ldots y(t-n_y);u(t-1),\ldots \mu(t-n_u)],\ldots \mu(t-n_u)] \tag{6.23}$$

where $I \in R^L; L = n_y + n_u$. Normally, n_y and n_u are, strictly speaking, unknown. In most cases it is practically sensible to select small numbers for e.g., 2 or 3 say. The output of the hidden layer is then constructed as follows:

$$O_j(t) = \frac{e^{\left(\sum\limits_{p=1}^{L} W_{1jp} I_p - \beta_{1j}\right)} - e^{\left(-\sum\limits_{p=1}^{L} W_{1jp} I_p - \beta_{1j}\right)}}{e^{\left(L\sum\limits_{p=1}^{L} W_{1jp} I_p - \beta_{1j}\right)} + e^{\left(-\sum\limits_{p=1}^{L} W_{1jp} I_p - \beta_{1j}\right)}}; j = 1,\ldots, M \tag{6.24}$$

where $O \in R^M$, and the output of the overall network can be obtained by means of the equation:

$$\hat{f}_0(.,.) = \frac{e^{\left(\sum\limits_{j=1}^{M} W_{2ij} O_j - \beta_{2j}\right)} - e^{\left(-\sum\limits_{j=1}^{M} W_{2ij} O_j - \beta_{2i}\right)}}{e^{\left(\sum\limits_{j=1}^{M} W_{2ij} O_j - \beta_{2i}\right)} + e^{\left(-\sum\limits_{j=1}^{M} W_{2ij} O_j - \beta_{2i}\right)}}; i = 1 \tag{6.25}$$

in which $W_1 \in R^{LxM}$ and $\beta_1 \in R^M$ are the weights and thresholds between the input layer and the hidden layer respectively and $W_2 \in R$ and $\beta_2 \in R$ are the weights and the thresholds between the hidden layer and the output layer respectively. They can updated by the equation set [Chichochi, 1993].

$$W_{1jp}(t+1) = W_{1jp}(t) + \eta_{jp}\delta_i O_j(t);\ j = 1,\ldots, M;\ p = 1,\ldots, L \tag{6.26}$$

$$W_{2ij}(t+1) = W_{2ij}(t) + \eta_{ij}\delta_i O_j(t);\ j = 1,\ldots, M; \tag{6.27}$$

$$\beta_{1j}(t+1) = \beta_{1j}(t) + \lambda_j \delta_{1j};\ j = 1,\ldots, M; \tag{6.28}$$

$$\beta_{2i}(t+1) = \beta_{2i}(t) + \lambda_i \delta_{2i};\ j = 1 \tag{6.29}$$

The training errors for this set are found simply from:

$$\delta_{2i} = \hat{f}_0(.,.)[1 - \hat{f}_0(.,.)][f_0(.,.) - \hat{f}_0(.,.)];\ i = 1 \tag{6.30}$$

and

$$\delta_{1j} = O_j(t)[1 - O_j(t)][\delta_{2i} W_{2ij}(t);\ j = 1,\ldots, M;\ i = 1 \tag{6.31}$$

6.5 Implementation of Self-tuning control algorithm procedure

A self-tuning control method is employed recursively to obtain an up-to-date model of the linear submodel. The algorithm at each time instant t can be outlined as consisting of the following steps (See Fig.3 for flow chart).

I. Sample the plant output *y(t)* and establish the data vectors $\varphi(t)$ and *I(t)* by means of the plant input *u(t)* and output *y(t)* data sequences.

II. Use the recursive algorithm from equations (6.17) and (6.18) to estimate the parameters of $\hat{A}(z^{-1})$ and $\hat{B}(z^{-1})$.

III. Calculate the controller parameters from (6.11) to (6.13)

IV. Obtain the controller output *u(t)* through (6.8)

V. Generate the next step training signal from (6.22)

VI. Train the BP network for a pre-selected number of times N, using (6.23) to (6.31).

VII. Wait for the sampling clock pulse then go to step I.

Note: Stability analysis of the control algorithm require complex mathematical derivation hence it is not pursued further here, however algorithms have been investigated elsewhere [Zhu *et al.* 1999].

6.6 Multivariable decoupling control algorithm

The single variate self-tuning control algorithm developed earlier has been expanded for multivariable non-linear systems, which exhibit coupling effects. Consider the following representation for n input and n output non-linear multivariable plants.

$$Y(t+1) = F(Y(t), U(t)) \tag{6.32}$$

where $F(Y,U) :\rightarrow R^{n}$; $\{Y \in R^{n_y}\}$ and $U \in R^{n_u}; n = n_y + n_u\}$ are a complicated non-linear mapping vectors and a smooth non-linear function vector respectively (i.e., infinitely differentiable). If it was the case with SISO plants (6.32) can be described by a generalised parametric model:

$$A(z^{-1})Y(t+1) = B(z^{-1})U(t) + F_0(.,.) \tag{6.33}$$

in which $A(z^{-1})$ and $B(z^{-1})$ are diagonal matrices such that:

$$\begin{aligned} A(z^{-1}) &= diag(1 + a_l^{ii} z^{-1} + \ldots + a_l^{ii} z^{-1}) \\ B(z^{-1}) &= diag(b_0^{ii} + b_l^{ii} z^{-1} + \ldots + b_m^{ii} z^{-m}); i = 1, \ldots, n \end{aligned} \tag{6.34}$$

The equivalent linear and sub model is originated by assuming that no coupling relationships exist. The coupling effects and the other non-linear relationships are meanwhile accommodated in $F_0(Y,U) :\rightarrow R^{n}$; $\{Y \in R^{n_y}\}$ and $U \in R^{n_u}; n = n_y + n_u\}$. The performance index can be defined as:

$$J = \|E(t+1)\|^2 = \left\|S(z^{-1})Y(t+1) - RD(t) - Q(z^{-1})u(t) - HF_0(.,.)\right\|^2 \tag{6.35}$$

where $D(t)$ is a bounded reference input vector and $S(z^{-1}), Q(z^{-1})$, R and H are diagonal weighting polynomial matrices. They are defined as:

$$\begin{aligned} S(z^{-1}) &= diag(1 + \xi_1^{ii} z^{-1} + \ldots + \xi_{n\xi}^{ii} z^{-n\xi}) \\ D(z^{-1}) &= diag(q_0^{ii} + q_1^{ii} z^{-1} + \ldots + q_{nq}^{ii} z^{-nq}); \\ R &= diag(r^{ii}); H = diag(h^{ii}); i = 1, \ldots, n \end{aligned} \tag{6.36}$$

Now we can define an auxiliary output as:

$$\phi(t+1) = S(z^{-1})Y(t+1) \tag{6.37}$$

In a similar fashion to the single variate case the optimal predictor $\phi^*(t+1)/t)$ for the auxiliary output $\phi(t+1)$ is described by:

$$\phi^*(t+1)/t) = G(z^{-1})Y(t+1) + K(z^{-1})B(z^{-1})U(t) + K(z^{-1})F_0(.,.) \tag{6.38}$$

where $G(z^{-1})$ and $K(z^{-1})$ are diagonal polynomial matrices and defined as:

$$\begin{aligned} G(z^{-1}) &= G_0^{ii} + g_1^{ii}z^{-1} + ... + g_{ng}^{ii}z^{-ng}); \\ K(z^{-1}) &= diag(1 + k_1^{ii}z^{-1} + ... + k_{nk}^{ii}z^{-nk}); i = 1,..., n \end{aligned} \tag{6.39}$$

which satisfy

$$S(z^{-1}) = K(z^{-1})A(z^{-1}) + z^{-1}G(z^{-1}); n_g = n - 1 \tag{6.40}$$

The control law, which minimises the performance index of (61), can then be found from:

$$\phi^*(t+1)/t) = RD(t) + Q(z^{-1})U(t) + HF_0(.,.) \tag{6.41}$$

in exactly the same way as for the single variate case, arriving at a similar form to (6.8) as a solution.

Substituting for the input signal, the closed loop equation of (6.33) is given by:

$$[B(z^{-1})S(z^{-1}) + Q(z^{-1})A(z^{-1})]Y(t+1) = B(z^{-1})RD(t) + [Q(z^{-1}) - B(z^{-1})H]F_0(.,.) \tag{6.42}$$

To produce satisfactory dynamics, a stable polynomial matrix $T(z^{-1})$ is chosen as:

$$T(z^{-1}) = diag(t_0^{ii} + t_1^{ii}z^{-1} + ... + t_{nt}^{ii}z^{-nt}); i = 1,..., n \tag{6.43}$$

and the following equation must be satisfy by solving for matrix polynomials, $\Xi(z^{-1})$ and $Q(z^{-1})$ in:

$$[B(z^{-1})S(z^{-1}) + Q(z^{-1})A(z^{-1}) = T(z^{-1}); n\xi = n-1; n_q = n-1; n_t \leq 2n-1 \tag{6.44}$$

To cancel the static offset, R can be chosen as:

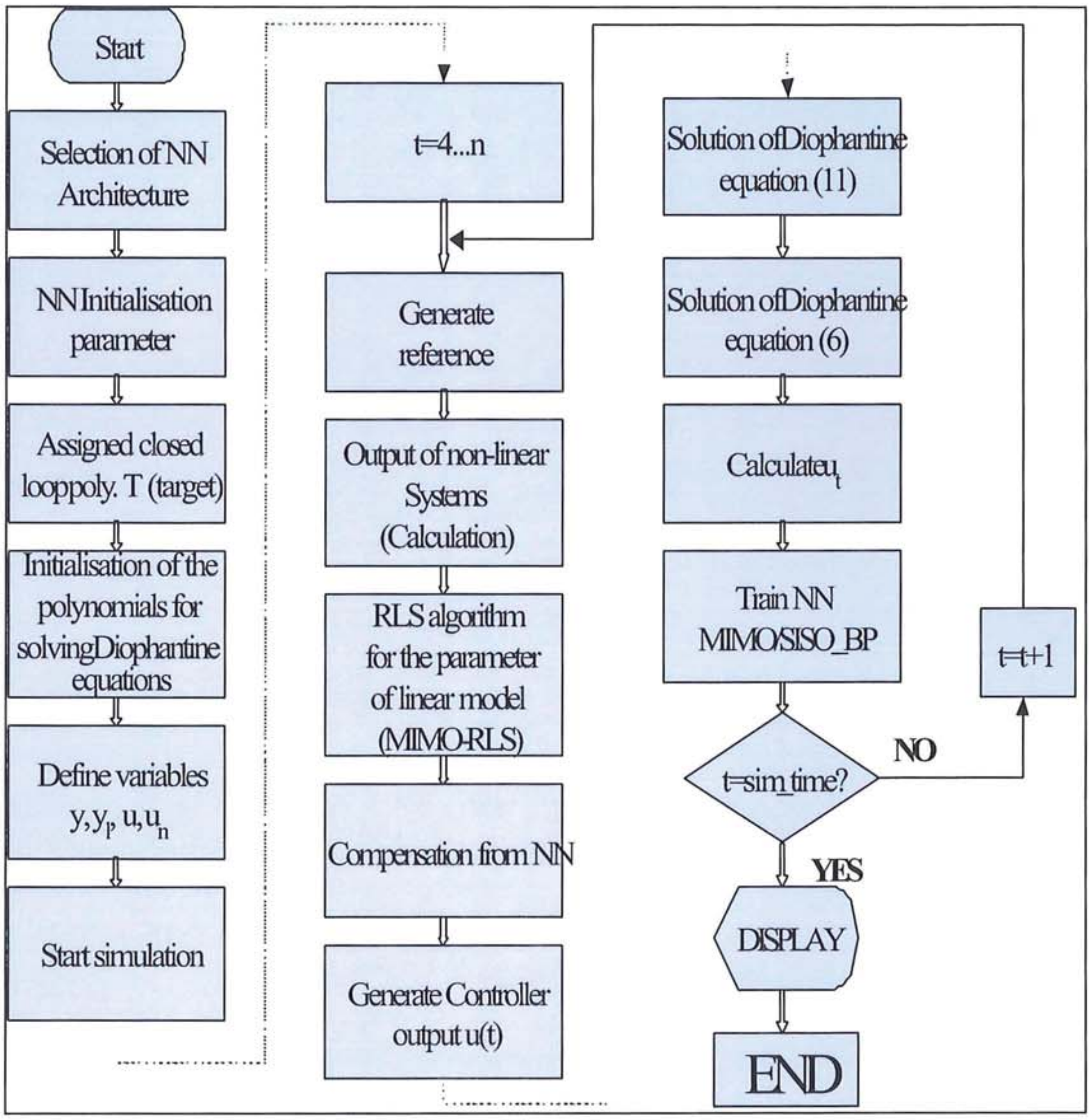
$$R^{ii} = \frac{T^{ii}(1)}{B^{ii}(1)}; i = 1,..., n \tag{6.45}$$

Finally, to eliminate the effect of non-linear and coupling actions, H is chosen by:

$$H^{ii} = \frac{Q^{ii}(1)}{B^{ii}(1)}; i = 1,..., n \tag{6.46}$$

It can be seen therefore, that in this type of controller, a multivariable requirement only presents a problem in terms of extra computational requirements, nothing more. The linear sub-model in (6.33) is initiated without regard to coupling effects; therefore the unknown linear parameters of $A(z^{-1})$ and $B(z^{-1})$ can be separately identified by a recursive algorithm. A neural network can then be used to approximate the non-linear function including any coupling effects. The main difference of the overall controller when compared with the SISO algorithm is that the neural network must be structured with n sigmoidal neurones in its output layer.

By disregarding coupling effects, a MIMO plant with n outputs can be viewed as n independent SISO plants, the self-tuning control algorithm presented based on SISO plants are therefore still applicable in the MIMO case as is considered here.



Start
Selection of NN Architecture
NN Initialisation parameter
Assigned closed looppoly. T (target)
Initialisation of the polynomials for solving Diophantine equations
Define variables y, y_p, u, u_n
Start simulation
t=4...n
Generate reference
Output of non-linear Systems (Calculation)
RLS algorithm for the parameter of linear model (MIMO-RLS)
Compensation from NN
Generate Controller output u(t)
Solution of Diophantine equation (11)
Solution of Diophantine equation (6)
Calculate u_t
Train NN MIMO/SISO_BP
t=sim_time?
NO
YES
t=t+1
DISPLAY
END

Fig.6.3: Flow chart for self-tuning algorithm

6.7 SIMULATION STUDIES

Simulation studies have been carried out to investigate a nonminimum phase, non-linear plant with proposed algorithm.

Example 1: A nonminimum phase, non-linear plant was investigated of the following form [Wellstead and Zarrop 1991]

$$y(k)=\frac{1.5\sin[y(k-1)]y(k-1)}{1+y(k-1)^2+y(k-2)^2}+1.1y(k-1)+1.2u(k-1)+2u(k-2) \qquad (1a)$$

The equivalent model was selected as:

$$(1+a_1z^{-1})y(k+1)=(b_0+b_1z^{-1})u(k)+f_0(.,.) \qquad (1b)$$

and desired dynamic polynomial $T(z^{-1})$ was selected as:

$$T(z^{-1})=1+0.5z^{-1}. \qquad (1c)$$

where k is the time step.

The simulation network parameters was set up as shown in Table 6.1 below.

Network Structure	**Adjustable Parameters**	**Figures**
Covariance matrix P_{ini} (RLS initialised as)	10000,	Fig.6.4 - Fig.6.7
Input layer (L)	8	
Hidden layer (H)	9	
Output layer (I)	1	
Learning rate (η)	0.02 (0.001-0.04)	
Momentum Constant alpha (α)	0.4 (see comment)	
Number of iteration	3 (best range 1-4)	
Neural network for predicting error set up n_i; n_h; and n	15;1 and 2	

Table 6.1: Parameter meter structure of example 1.

The network was trained 3 times in each sampling period. From the controlled system simulation output results (Fig.6.4) along with the reference input signal shows that the above algorithms are quite satisfactory. The output values from the non-linear sub-model and the back-propagation network as shown in Fig.6.5, indicates that the non-linear sub-model of the plant can be satisfactorily tracked by using BP neural network. Fig.6.6 represents the results of estimated linear sub-model parameter and Fig.6.7 is the controller output.

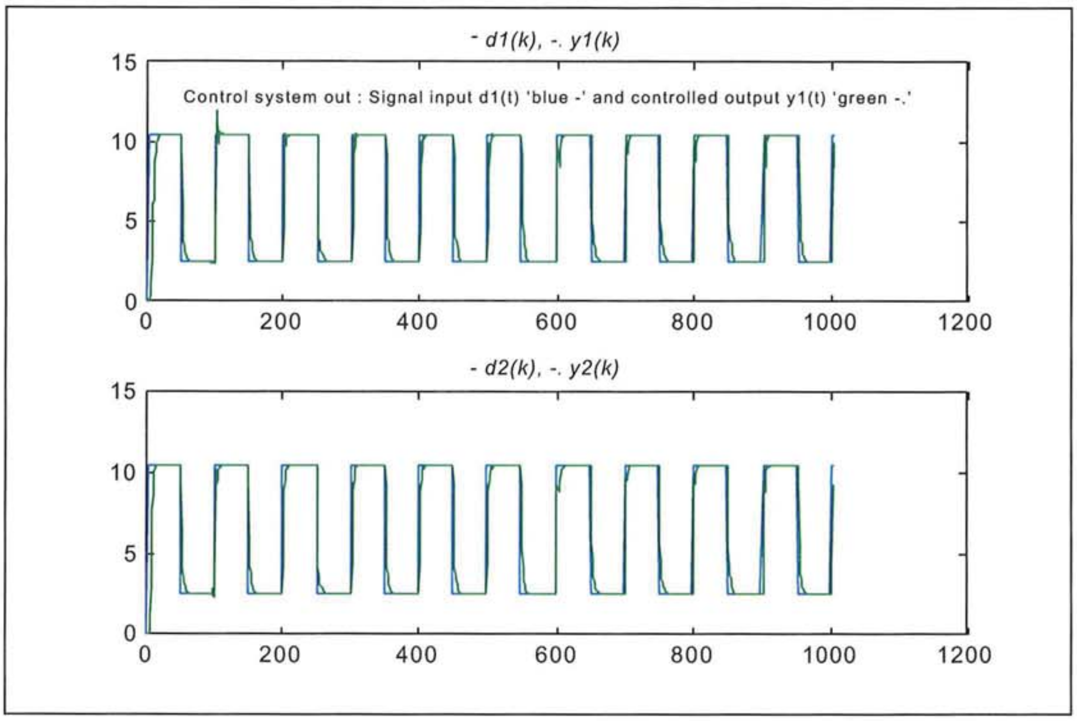


Fig.6.4 Controlled system output of example 1.

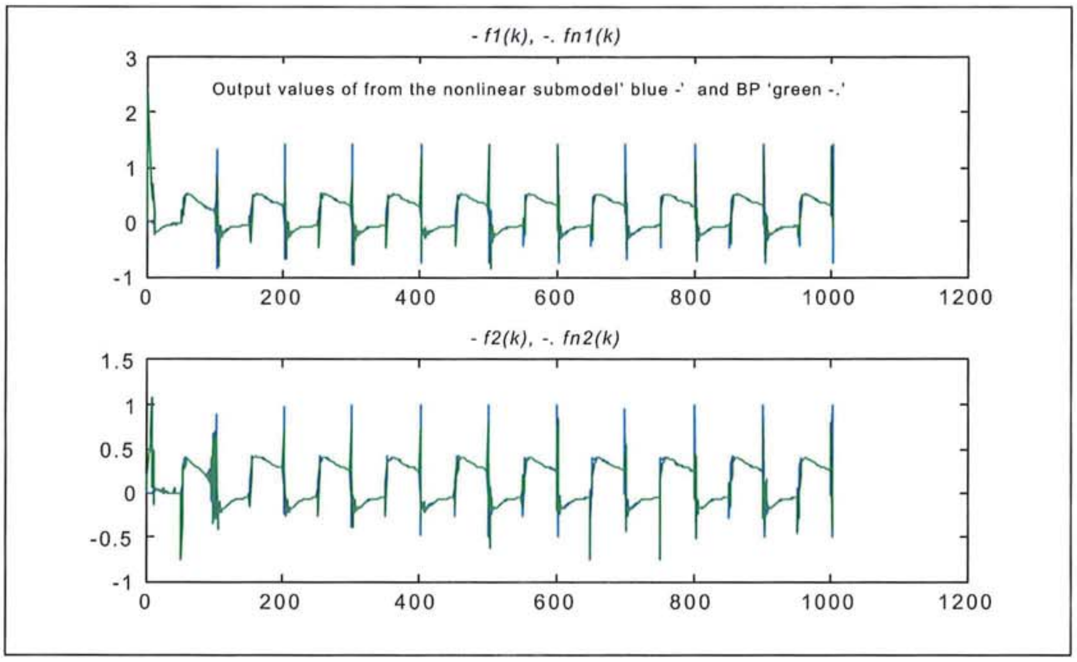


Fig.6.5 The output values from the nonlinear submodel and BP neural network of example 1.

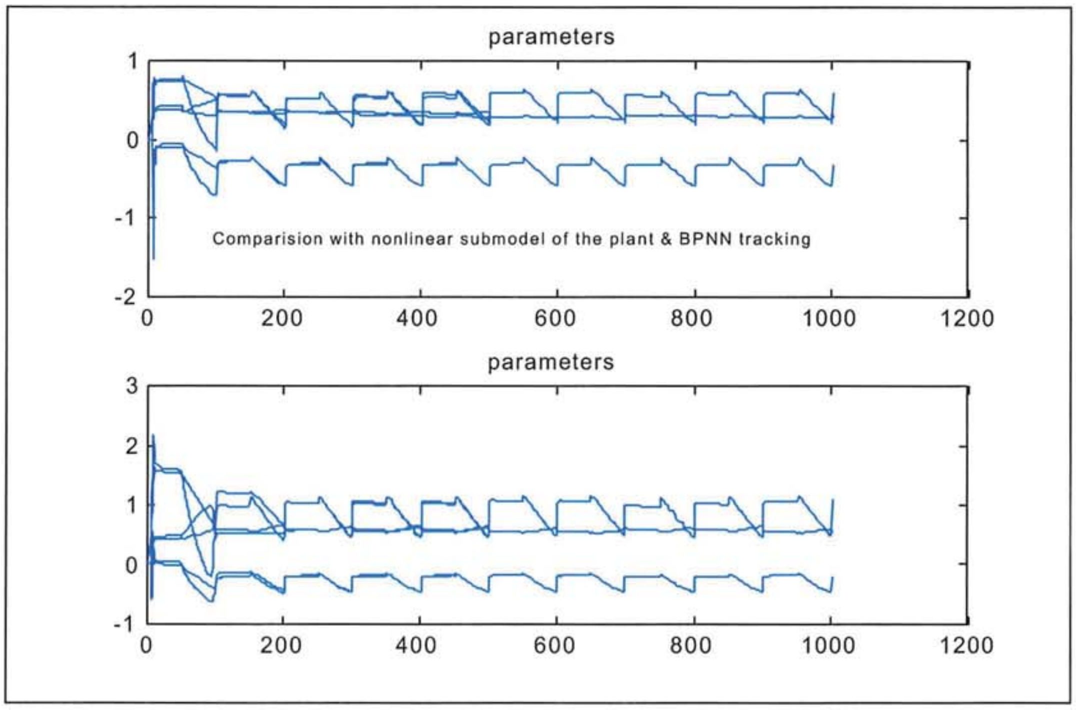


Fig.6.6. Results of estimated linear submodel paramters of example 1.

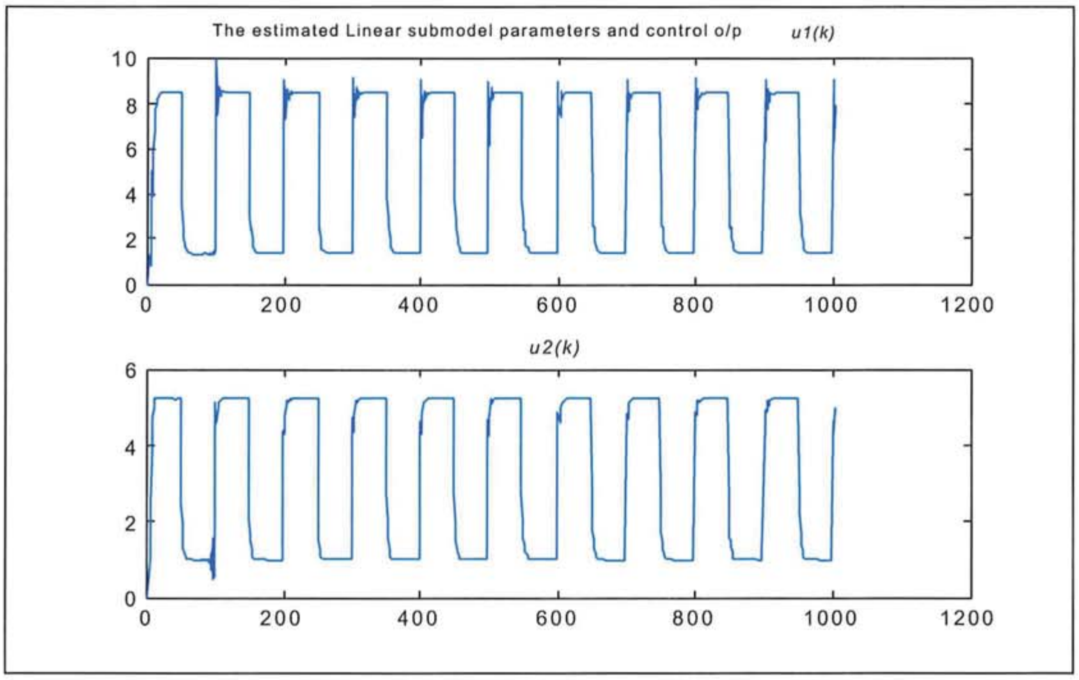


Fig.6.7 Controller output of example 1

Simulation 2: The following multivariable non-linear coupled system was investigated:

$$y_1(k) = \frac{2y_1(k-1)y_1(k-2)}{1 + y_1(k-1)y_2(k-1) + y_1(k-2)^2} + 0.3u_1(k-1) + 0.7u_1(k-2) + 0.2u_1(k-2);$$

$$y_2(k) = \frac{1.5y_2(k-1)y_2(k-2)}{1 + y_2(k-1)^2 y_1(k-2) + y_2(k-2)} + 0.5u_2(k-1) + 1.2u_2(k-2) + 0.1u_1(k-2);$$

(2a)

The equivalent model was chosen as:

$$A(z^{-1})Y(k) = B(z^{-1})U(k) + F_0(.,.) \tag{2b}$$

where

$$A(z^{-1}) = diag(1 + a_1^{ii}); \quad B(z^{-1}) = diag(b_0^{ii} + b_0^{ii} z^{-1}); \tag{2c}$$

where $i = 1,2,...n$

in this case the desired dynamical polynomial $T(z^{-1})$ was chosen as

$$T(z^{-1}) = diag(1 + 0.5z^{-1}) \tag{2d}$$

In this simulation experiment, the neural network structure and parameters were set up as shown in Table 6.2.

Network Structure	**Adjustable Parameters**	**Figures**
Covariance matrix P_{ini} (RLS initialised as)	10000,	Fig.6.8 - Fig.6.11
Input layer (L)	8	
Hidden layer (H)	9	
Output layer (I)	2	
Learning rate (η)	0.02 (0.001-0.04)	
Momentum Constant alpha (α)	0.4 (see comment)	
Number of iteration	3 (best range 1-4)	
Neural network for predicting error set up n_i; n_h; and n	15;1 and 2	

Table 6.2: Parameter structure of example 2.

The network was trained three times in each sample period. The controlled system outputs are shown in Fig.6.8. They show a good tracking of reference signal, which imply that the respectable decoupling control is possible to be achieved. The output values from the non-linear submodel and the BP network and the estimated linear submodel parameters are shown in Fig.6.9 and Fig.6.10 respectively whilst the controller output is shown in Fig.6.11.

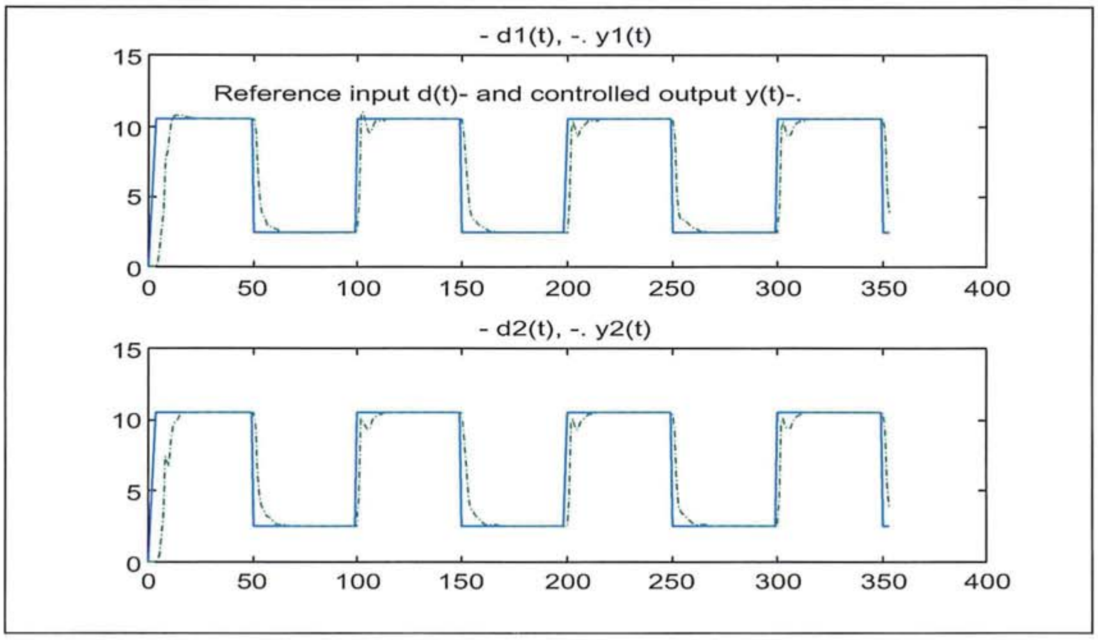


Fig.6.8: Controlled system output of example 2.

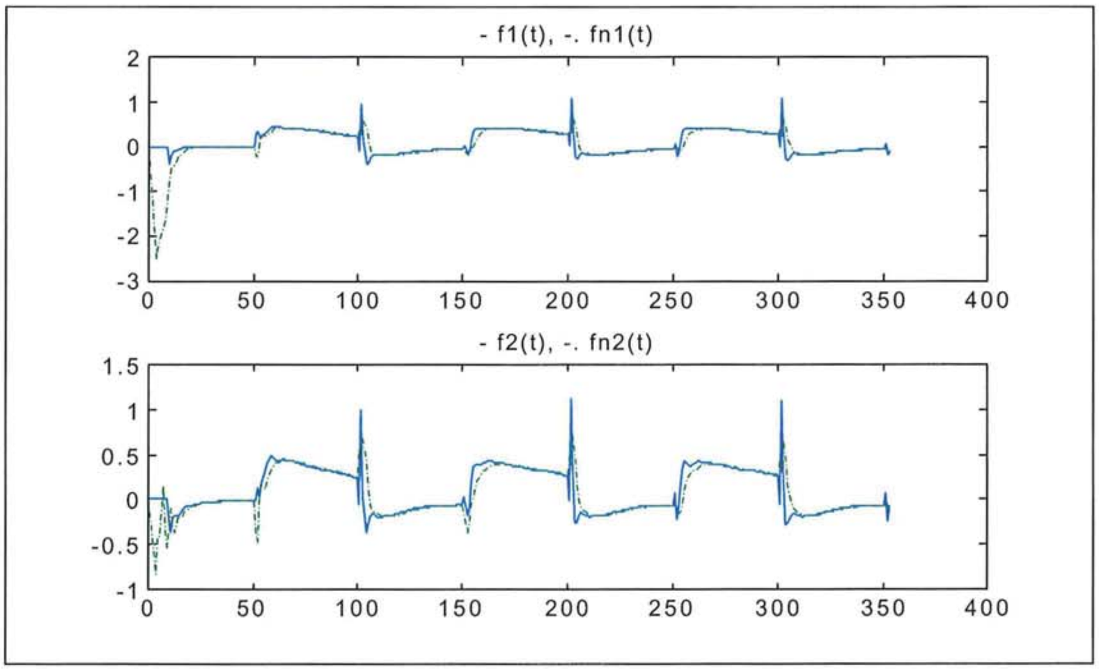


Fig.6.9: Output values from the nonlinear submodel and the BP neural network of example 2

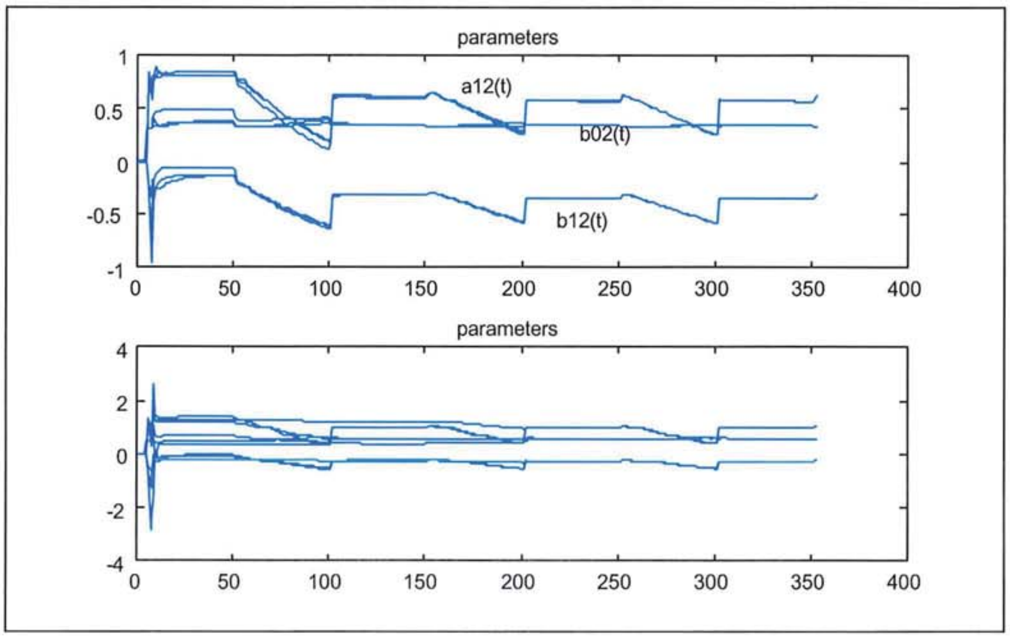


Fig.6.10: Results of estimated linear submodel paramters of example 2

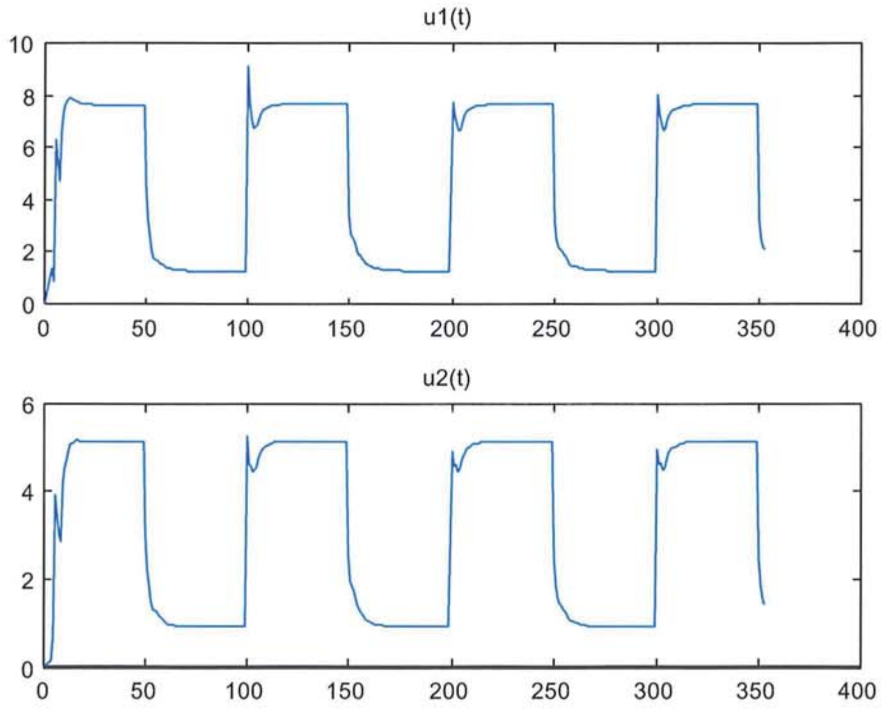


Fig.6.11. Controller output of example 2.

6.8 CONCLUSIONS

This chapter has introduced the way in which complex non-linear control system problems can be represented as a linear system along with a non-linear part. This approach seems to eliminate unwanted elements in the system. Here an adaptive control strategy is applied to linear systems and this has been modified to accommodate a non-linear neural network sub-model. This sub-model, is a backpropagation network and is applied for approximating the modelling errors such as those caused by non-linearities, uncertainties and coupling effects.

The self-tuning adaptive control is traditionally limited to unknown linear systems. By introducing back-propagation neural networks into the self-tuning scheme, it is demonstrated that this new technique has the potential to deal with unknown linearizable non-linear systems.

The main advantages and disadvantages of this non-linear modelling design approach are:

Advantages:

- mathematical problems ordinarily associated with an overall non-linear design can be avoided
- computer implementation effort for on-line real-time can be reduced significantly.

Disadvantages

- Learning can be slow if sampling time is too long, however, it can be overcome by updating algorithm or using fast processor

Remarks from simulation observation

- Successful parameter estimation can be performed with the aid of forgetting a factor of less than unity, but only if an appropriately changing input is present.
- In practice we have to fix the size of the neural network before it is used in the adaptive control system. Thus, the modelling error $\in > 0$ is determined.
- A lesson to be learned is that a self-tuning controller must incorporate a check for unreasonable values of the estimated parameters within its software, so as to avoid blow up.
- This minimum variance of self-tuning adaptive controller may be limited in the real practical environment due to noise actuator and output constraints etc.

CHAPTER 7: DYNAMICALLY DRIVEN RECURRENT NEURAL NETWORKS

7.1 INTRODUCTION

Recurrent networks are neural networks with one or more feedback loops. The feedback can be of a *local* or *global* type. The objective of this chapter is to study recurrent neural networks with global feedback.

Given a mulitlayer perceptron as the basic building block, the application of global feedback can take a variety of forms. We may have feedback from the output neurons of the multilayer perceptron to the input layer. Yet another possible form of global feedback is from the hidden neurons of the network to the input layer. When the multilayer perceptron has two or more hidden layers, the possible forms of global feedback expand even further. The point is that recurrent neural networks have a rich repertoire of architectural layouts.

Basically there are two functional uses of recurrent networks, which are:

- a*ssociative memories* and
- *Input-output mapping networks.*

The use of recurrent networks as associative memories is found in [Haykin, 1999 Chp.14]. However the aim of this chapter is to study the input-output mapping networks in the state space form.

By definition, the input space of a mapping network is mapped onto an output space. For this kind of application, a recurrent network responds *temporally* to an externally applied input signal hence Haykin calls this type of network is *"dynamically driven recurrent neural network"*. The application of feedback enables recurrent networks to acquire *state* representations, which make them suitable devices for diverse applications for example non-linear prediction, modelling, adaptive equalisation of communication channels, speech processing, plant control, and automobile engine diagnostics. As such, recurrent networks offer an alternative to the dynamically driven feed-forward networks.

Because of the beneficial effects of global feedback, they may actually fare better in these applications. The use of global feedback has the potential of reducing memory requirement significantly.

Organisation of the Chapter

The chapter is organised in four parts: architectures, theory, learning algorithms and applications.

7.2 RECURRENT NETWORK ARCHITECTURES

As mentioned in the introduction, the architectural layout of a recurrent network takes many different forms.

This section describes three specific network architectures, each of which highlights a specific form of global feedback. They share the following common features:

- They all incorporate a static multilayer perceptron or parts thereof.
- They all exploit the non-linear mapping capability of the multilayer perceptron.

7.2.1 Input-Output Recurrent Model

The architecture of a generic recurrent network is shown in Fig.7.1. It is naturally derived from a multilayer perceptron. The model has a single input that is applied to a tapped-delay-line (TDL) memory of q units. It has a single output that is fed back to the input via another TDL memory also of q units. The contents of these two TDL memories are used to feed the input layer of the multilayer perceptron. The present value of the model input is denoted by $u(n)$, and the corresponding value of the model output is denoted by $y(n+1)$; that is, the output is ahead of the input by one time unit. Thus, the signal vector applied to the input layer of the multilayer perceptron consists of a data window made up as follows:

- Present and past values of the input, namely $u(n)$, $u(n-1),\ldots,u(n-q+1)$ which represents *exogenous* inputs originating from outside the network.
- Delayed values of the output, namely, $y(n)$, $y(n-1),\ldots$, $y(n-q+1)$, on which the model output $y(n+1)$ is *regressed*.

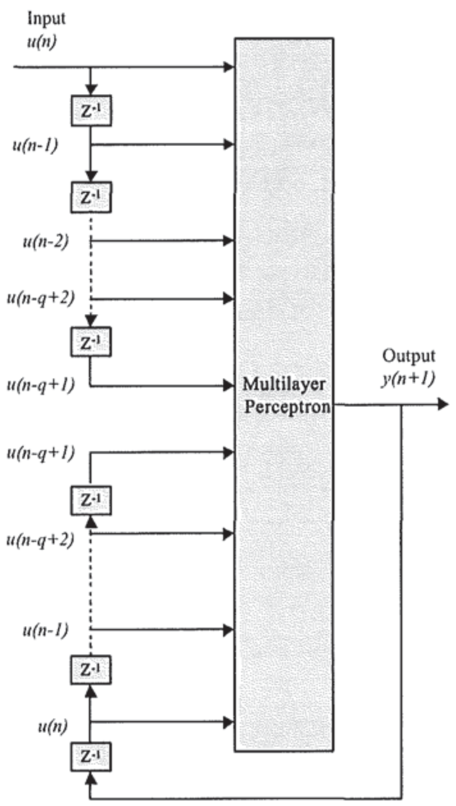


Fig.7.1 NARX inputs model.

Thus, the recurrent network of Fig. 7.1 is referred to as a *non-linear auto-regressive with exogenous inputs (NARX) model.* The dynamic behaviour of the NARX model is described by

$$y(n+1) = F(y(n),..., y(n-q+1), u(n),..., u(n-q+1)) \tag{7.1}$$

where *F(.)* is a non-linear function of its arguments. Note that in Fig.7.1 assumed that the two delay-line memories in the model are both of size q; they are generally different. The NARX model is further extended in Section 7.4.

7.2.2 State-Space Model

Fig.7.2 shows the block diagram of another generic recurrent network, called a *state-space* model. The hidden neurones define the *state* of the network. The output of the hidden layer is fed back to the input layer via a bank of unit delays. The input layer consists of a concatenation (connected chain) of feedback nodes and source nodes. The network is connected to the external environment via the source nodes. The number of unit delays used to feed the output of the hidden layer back to the input layer determines the order of the model. Let the m-by-1 vector $u(n)$ denote the input vector, and the q-by-l vector $\boldsymbol{x}(n)$ denote the output of the hidden layer at time n. The dynamic behaviour of the model in Fig.7.2 can be written by the pair of coupled equations:

$$x(n+1) = f(x(n), u(n)) \tag{7.2}$$

$$y(n) = Cx(n) \tag{7.3}$$

where $f(\cdot,\cdot)$ is a non-linear function characterising the hidden layer, and C is the matrix of synaptic weights characterising the output layer. The hidden layer is non-linear, but the output layer is linear.

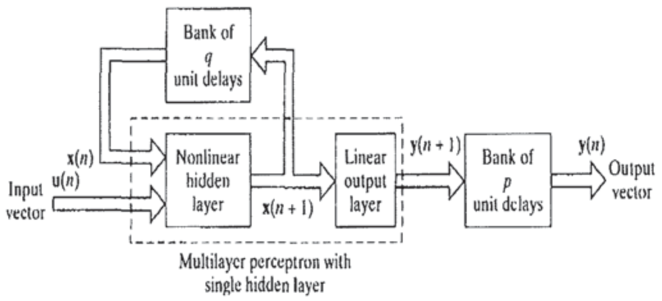


Fig.7.2 State-space model

The recurrent network of Fig.7.2 includes several recurrent architectures as special cases. Consider, for example, the *simple recurrent network (SRN)* described in [Elman, 1990] and depicted in Fig.7.3. Elman's network has architecture similar to that of Fig. 7.2 except for the fact that the output layer may be non-linear and the bank of unit delays at the output is omitted.

Elman's network contains recurrent connections from the hidden neurones to a layer of *context units* consisting of unit delays. These context units store the outputs of the hidden neurones for one time step, and then feed them back to the input layer. The hidden neurones thus have some record of their prior activations, which enables the network to perform learning tasks that extend over time. The hidden neurones also feed the output neurones that report the response of the network to the externally applied stimulus. Due to the nature of feedback around the hidden neurones, these neurones may continue to recycle information through the network over multiple time steps, and thereby discover abstract representations of time. The simple recurrent network is therefore not merely a tape recording of past data.

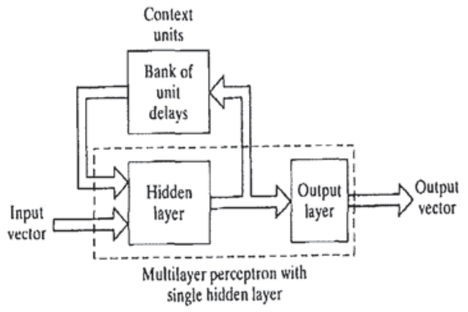


Fig.7.3 Simple Recurrent Network (SRN)

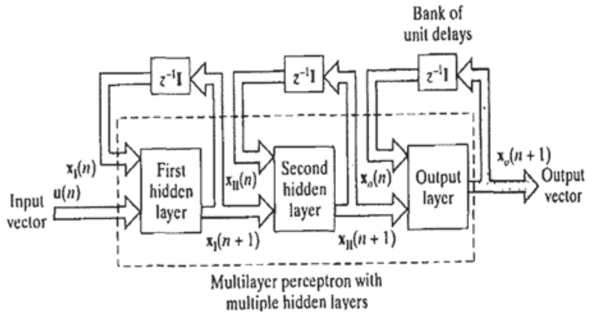


Fig.7.4 Recurrent multilayer perceptron.

Elman 1990 discusses the use of the simple recurrent network shown in Fig.7.3 to discover word boundaries in a continuous stream of phonemes without any built-in representational constraints. The input to the network represents the current phoneme. The output represents the network's best guess as to what the next phoneme is in the sequence. The role of the context units is to provide the network with *dynamic memory* to encode the information contained in the sequence of phonemes, which is relevant to the prediction.

7.2.3 Recurrent Multilayer Perceptron

The third recurrent architecture considered here is known as a *recurrent multilayer perceptron (RMLP)* [Puskorius *et al.*, 1996]. It has one or more hidden layers, for the same reasons that static multilayer perceptrons are often more effective and parsimonious (sparing) than those using a single hidden layer. Each computation layer of an RMLP has feedback around it, as illustrated in Fig.7.4 for the case of an RMLP with two hidden layers.

Let the vector $\boldsymbol{x}_I(n)$ denote the output of the first hidden layer, $\boldsymbol{x}_{II}(n)$ denote the output of the second hidden layer and so on. Let the vector $\boldsymbol{x}_0(n)$ denote the output of the output layer. Then the dynamic behaviour of the RMLP, in general, in response to an input vector $\boldsymbol{u}(n)$ is described by the following system of coupled equations:

$$
\begin{aligned}
&x_I(n+1) = \varphi_I(x_I(n), u(n)) \\
&x_{II}(n+1) = \varphi_{II}(x_{II}(n), x_I(n+1)) \\
&\vdots \\
&x_0(n+1) = \varphi_0(x_0(n), x_h(n+1))
\end{aligned}
\qquad (7.4)
$$

where $\varphi_I(.,.)$, $\varphi_{II}(.,.)$, $\varphi_0(.,.)$ denote the activation functions characterising the first hidden layer, second hidden layer,..., and output layer of the RMLP, respectively and h denotes the number of hidden layers in the network.

The RMLP described herein subsumes the Elman network of Fig.7.3 and the state-space model of Fig.7.2 since the output layer of the RMLP or any of its hidden layers is not constrained to have a particular form of activation function.

7.3 STATE-SPACE MODEL

The notion of *state* plays a vital role in the mathematical formulation of a dynamical system. The state of a dynamical system is formally defined as a *set of quantities that summarises all the information about the past behaviour of the system that is needed to uniquely described its future behaviour, except for the purely external effects arising from the applied input (excitation)* [Haykin, 1999]. Let the q-by-1 vector x *(n)* denote the state of a non-linear discrete-time system. Let the m-by-1 vector $u(n)$ denote the input applied to the system and the p by-1 vector y *(n)* denote the corresponding output of the system. In mathematical terms, the dynamic behaviour of the system, assumed to be *noise free,* is described by the following pair of non-linear equations [Sontag, 1996]:

$$x(n+1) = \varphi(w_a x(n) + w_b u(n)) \qquad (7.5)$$

$$y(n) = Cx(n) \qquad (7.6)$$

where w_a is a q by q matrix, w_b is a q by $(m+1)$ matrix, $\boldsymbol{C}$ is a p-by-q matrix and $\varphi : \Re^q \to \Re^q$ is a diagonal map described by:

$$\varphi : \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{bmatrix} \to \begin{bmatrix} \varphi(x_1) \\ \varphi(x_2) \\ \vdots \\ \varphi(x_q) \end{bmatrix} \qquad (7.7)$$

for some memoryless, component-wise nonlinearity $\varphi : \Re^q \to \Re^q$. The spaces $\Re^m$, $\Re^q$ and $\Re^p$ are called the *input space, state space and output space,* respectively. The dimensionality of the state space, namely q, is the *order* of the system. Thus, the state space model Fig. 7.2 is an *m-input, p-output recurrent model of order q.* Equation (7.6) is the *measurement equation.* The process equation (7.5) is a special form of (7.7).

The recurrent network of Fig.7.2, based on the use of a static multilayer perceptron and two delay-line memories, provides a method for implementing the non-linear feedback system described by Eqs (7.5) to (7.7). Note that in Fig.7.2 *only those neurones in the multilayer perceptron that feed back their outputs to the input layer via delays are responsible for defining the state of the recurrent network.* This statement therefore excludes the neurones in the output layer from the definition of the state.

For the interpretation of matrices $\boldsymbol{W}_a$, $\boldsymbol{W}_b$ and $\boldsymbol{C}$ and non-linear function $\varphi(.)$ we may say:

- The matrix $\boldsymbol{W}_a$, represents the synaptic weights of the q neurones in the hidden layer that are connected to the feedback nodes in the input layer. The matrix $\boldsymbol{W}_b$ represents the synaptic weights of these hidden neurones that are connected to the source nodes in the input layer. It is assumed that the bias terms for the hidden neurones are absorbed in the weight matrix $\boldsymbol{W}_b$.
- The matrix $\boldsymbol{C}$ represents the synaptic weights of the p linear neurones in the output layer that are connected to the hidden neurones. It is assumed that the bias terms for the output neurones are absorbed in the weight matrix $\boldsymbol{C}$.
- The non-linear function $\varphi(.)$ represents the sigmoid activation function of a hidden neurone. The activation function typically takes the form of a hyperbolic tangent function:

$$\varphi(x) = \tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \qquad (7.8)$$

or a logistic function:

$$\varphi(x) = \frac{1}{1+e^{-x}} \qquad (7.9)$$

An important property of a recurrent network described by the state-space model of Eqs. (7.5) and (7.6) is that it can *approximate* a wide class of non-linear dynamical systems. However, the approximations are only valid on compact subsets of the state space and for finite time intervals, so that interesting dynamical characteristics are not reflected [Sontag, 1992].

Controllability and Observability

In the study of system theory, stability, controllability and observability are prominent features, each in its own fundamental way. Controllability and observability are treated together, in [Levin and Narendra, 1993; Haykin, 1999] therefore only some issues are highlighted below.

As mentioned earlier, many recurrent networks can be represented by the state-space model shown in Fig.7.2, where the state is defined by the output of the hidden layer fed back to the input layer via a set of unit delays. In that context, it is important to know, whether or not the recurrent network is controllable and observable. Controllability is concerned with whether or not we can control the dynamic behaviour of the recurrent network. Observability is concerned with whether or not we can observe the result of the control applied to the recurrent network. In that sense, observability is the dual of controllability.

A recurrent network is said to be *controllable* if an initial state is steerable to any desired state within a finite number of time steps; the output is irrelevant to this definition. The recurrent network is said to be *observable* if the state of the network can be determined from a finite set of input/output measurements.

7.4 NON-LINEAR AUTOREGRESSIVE WITH EXOGENOUS (NARX) INPUTS MODEL

Consider a recurrent network with a SISO, whose behaviour is described by the state

$$x(n+1) = \varphi(w_a x(n) + w_b u(n)) \tag{7.10}$$

$$y(n) = C^T x(n) \tag{7.11}$$

Eqs. (7.10) and (7.11). Given this state-space model, we wish to modify it into an input-output model as an equivalent representation of the recurrent network.

Using (7.10) and (7.11), we may show that the output $y(n+q)$ is expressible in terms of the state $x(n)$ and the vector of inputs $u_q(n)$ as follows:

$$y(n+q) = \phi(x(n), u_q(n)) \tag{7.12}$$

where q is the dimensionality of the state space, and $\phi : \Re^{2q} \rightarrow \Re$. Provided that the recurrent network is observable, we may use the local observability theorem to write

$$x(n) = \psi(y_q(n), u_{q-1}(n)) \tag{7.13}$$

where $\psi : \Re^{2q-1} \rightarrow \Re^{q}$. Hence, substituting (7.12) in (7.13) gives

$$\begin{aligned} y(n-q) &= \phi(\psi(y_q(n), u_{q-1}(n)), u_q(n)) \\ &= F(y_q(n), u_q(n)) \end{aligned} \tag{7.14}$$

where $u_{q-1}(n) = [u(n), u(n+1), \ldots u(n+q-2)]^T$;

$$y_q(n) = [y(n), y(n+1), \ldots y(n+q-1)]^T$$

where $u_{q-1}(n)$ is contained in $u_q(n)$ as its first (q^{-1}) elements and the non-linear mapping $F : \Re^{2q} \rightarrow \Re$ takes care of both ϕ and ψ. Using the definition of $y_q(n)$ and $u_q(n)$, we may rewrite (7.14) in the expanded form:

$$y(n+q) = F(y(n+q-1), \ldots, y(n), u(n+q-1), \ldots, u(n))$$

Replacing n with $n\text{-}q\text{+}1$, we may equivalently write [Narendra, 1995].

$$y(n+1) = F(y(n), \ldots, y(n-q+1), u(n), \ldots, y(n-q+1)) \tag{7.15}$$

In words, this can be expressed as some non-linear mapping $F : \Re^{2q} \rightarrow \Re$ exists whereby the present value of the output $y(n+1)$ is uniquely defined in terms of its past values $y(n), \ldots, y(n\text{-}q\text{+}1)$ and the present and past values of the input $u(n), \ldots, u(n\text{-}q\text{+}1)$. For this input-output representation to be equivalent to the state-space model of Eqs. (7.10) and (7.11). The recurrent network must be observable. The practical implication of this equivalence is that the NARX model of Fig.7.1, with its global feedback limited to the output neurone, is in fact able to simulate the corresponding fully connected recurrent state space model of Fig.7.2, (assuming that $m=1$ and $p=1$) with no difference between their input-output behaviour exist.

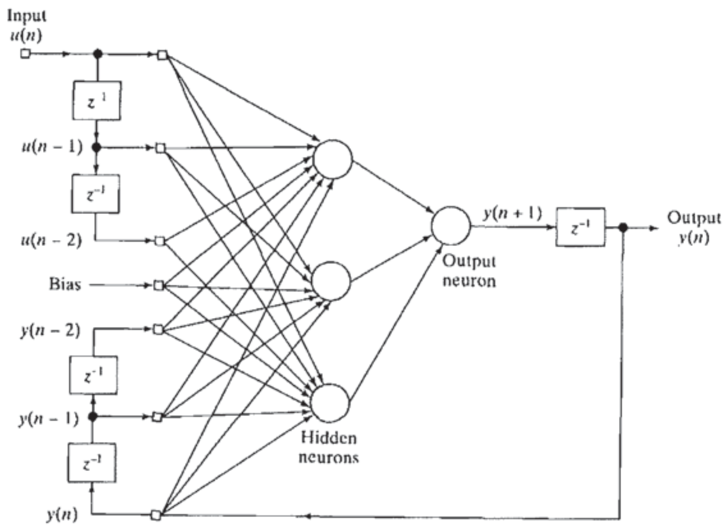


Fig. 7.5 NARX network with $q = 3$ hidden neurones.

7.5 LEARNING ALGORITHMS

The objective here is to highlight the issue of training recurrent networks. From earlier chapters we recall that there are two modes of training an ordinary (static) multilayer perceptron: batch mode (*off-line*) and sequential mode (*on-line*).

In the batch mode, the sensitivity of the network is computed for the entire training set before adjusting the free parameters of the network. In the sequential mode , on the other hand, parameter adjustments are made after the presentation of each pattern in the training set. Likewise, we have two modes of training a recurrent network, as described here [Williams and Zipser, 1995]:

1. ***Epochwise training***. For, a given epoch, the recurrent network starts running from some initial state until it reaches a new state, at which point the training is stopped and the network is reset to an initial state for the next epoch. The initial state doesn't have to be the same for each epoch of training. Rather, what is important is for the initial state for the new epoch to be different from the state reached by the network at the end of the previous epoch. Consider, for example, the use of a recurrent network to emulate the operation of a finite-state machine, that is, a device whose distinguishable internal configurations (states) are finite in number. In such a situation it is reasonable to use epochwise training since we have a good possibility for a number of distinct initial states and a set of distinct final states in the machine to be emulated by the recurrent network. In epochwise training for recurrent networks the term "epoch" is used in a sense different from that for an ordinary multilayer perceptron. In the current terminology, the epoch for the recurrent network corresponds to one training pattern for the ordinary multilayer perceptron.
2. ***Continuous training***. This second method of training is suitable for situations where there are no reset states available and / or on-line learning is required. The distinguishing feature of continuous training is that the network learns while signal processing is being performed by the network. Simply put, the learning process never stops. Consider, for example, the use of a recurrent network to model a non-stationary process such as a speech signal. In this kind of situation, continuous operation of the network offers no convenient times at which to stop the training and begin anew (fresh again) with different values for the free parameters of the network.

Keeping these two modes of training in mind, there are following learning algorithms exist [Haykin, 1999]:

- The BPTT algorithm, operates on the premise that the temporal operation of a recurrent network may be unfolded into a multi-layer perceptron. The *BPTT algorithm* for training a recurrent net work is an extension of the standard BP algorithm[2]. It may be derived by *unfolding* the temporal operation of the network into a layered feed-forward network, the topology of which grows by one layer at every time step.
- The real-time recurrent learning algorithm is derived from the state-space model described by Eqs. (7.5) and (7.6).

[2] The idea behind BPTT is that for every recurrent network it is possible to construct a feed-forward network with identical behaviour over a particular time interval (Minsky and Papert, 1969). BPTT was first describe in the Ph.D thesis of Werbos (1974); see also Webos (1990). The algorithm was rediscovered independently by Rumelhart *et al.*. (1986b). A variant of the BPTT algorithm is described in Williams and Peng (1990). For a review of the algorithm and related issues see Williams and Zipser (1995).

These two algorithms share many common features. First, they are both based on the method of gradient descent whereby the instantaneous value of a cost function (based on a squared-error criterion) is minimised w.r.t the synaptic weights of the network. Second, they are both relatively simple to implement, but can be slow to converge. Third, they are related in that the signal-flow graph representation of the back-propagation-through-time algorithm can be obtained from *transposition* of the signal-now graph representation of a certain form of the real-time recurrent learning [Haykin, 1999].

Real-time (continuous) learning, based on gradient descent, uses the *minimum amount of available information*, namely an instantaneous estimate of the gradient of the cost function with respect to the parameter vector to be adjusted. This may be accelerate the learning process by exploiting Kalman filter theory which utilises information contained in the training data more effectively [Haykin, 1999].

Heuristics Learning [Haykin, 1999]

- Heuristics training of recurrent networks that involve the use of gradient-descent methods.
- The training should begin with a small training sample, and then its size should be incrementally increased as the training proceeds.
- The synaptic weights of the network should be updated only if the absolute error on the training sample currently being processed by the network is greater than some prescribed criterion.
- The use of weight decay during training is recommended; weight decay, a crude form of complexity regularisation.

The first heuristic is of particular interest. If implementable, it provides a procedure for alleviating the vanishing gradient problem that arises in recurrent networks trained by means of gradient-descent methods.

7.6 System Identification

System identification is the experimental approach to the modelling of a process or a plant of unknown parameters [3]. It involves the following steps. (Also see Chapter 4):

(i) experimental planning, (ii) the selection of a model structure, (iii) parameter estimation and model validation. The procedure of system identification, as pursued in practice, is iterative in nature in that we may have to go back and forth between these steps until a satisfactory model is built.

Suppose then we have an unknown non-linear dynamical plant, and the requirement is to build a suitably parameterised identification model for it. We have the choice of basing the identification procedure on a *state-*

space model or *an input-output model*. The decision as to which of these two representations are used hinges on prior information of the inputs and observable of the system. In what follows, both representations are discussed.

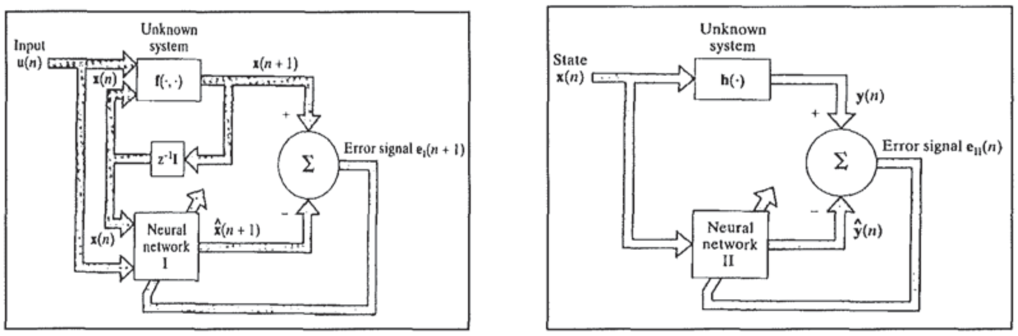
7.6.1 System Identification Using the State-Space Model

Suppose that the given plant is described by the state-space model:

$$\mathbf{x}(n+1)=f(\mathbf{x}(n),\mathbf{u}(n)) \tag{7.16}$$

$$\mathbf{y}(n)=\mathbf{h}(\mathbf{x},\mathbf{u}(n)) \tag{7.17}$$

where $f(\cdot,\cdot)$ and $h(\cdot)$ are vector-valued non-linear functions, both of which are assumed to be unknown; (7.17) is a generalisation of (7.6). We use two neural networks to identify the system, one for dealing with the process equation (7.16) and the other for dealing with the measurement equation (7.17), as shown in Fig. 7.6.



(a) (b)

Fig. 7.6 State space solution for the system identification problem.

We recognise that the state $x(n)$ is the one-step delayed version of $x(n+1)$. Let $\hat{x}(n+1)$ denote the estimate of $x(n+1)$ produced by the first neural network, labelled network I in Fig.7.6a. This network operates on a concatenated input consisting of the external input $u(n)$ and the state $x(n)$ to produce $\hat{x}(n+1)$. The estimate $\hat{x}(n+1)$ is subtracted from the actual state $x(n+1)$ to produce the error vector

$$e_I(n+1)=x(n+1)-\hat{x}(n+1)$$

where $x(n+1)$ plays the role of desired response. It is assumed that the actual state $x(n)$ is physically accessible for it to be used in this way. The error vector $e_I(n+1)$ is in turn used to adjust the synaptic weights of neural

* System identification has an extensive literature. For a treatment of the subject in book form, see Ljung (1987 & 1999), P.Norton (1987) and Ljung and Glad (1994). For an overview of the subject with an emphasis on neural networks, see Sjoberg *et al.*. (1995) and Narendra and Parthasarathy (1990).

network *I*, as indicated in Fig. 7.6 (a), so as to minimise the cost function based on the error vector $e_I(n + l)$ in some statistical sense.

The second neural network, labelled network *II* in Fig. 7.6(b), operates on the actual state $x(n)$ of the unknown plant to produce an estimate $\hat{y}(n)$ of the actual output $y(n)$. The estimate $\hat{y}(n)$ is subtracted from $y(n)$ to produce the second error vector

$$e_{II}(n) = y(n) - \hat{y}(n)$$

where $y(n)$ plays the role of desired response. The error vector $e_{II}(n)$ is then used to the Euclidean norm of the error adjust the synaptic weights of network II to minimise vector $e_{II}(n)$ in some statistical sense.

The two neural networks shown in Fig.7.6 operate in a synchronous fashion to provide a state-space solution to the system identification problem [Narendra and Parthasarathy, 1990]. Such a model is referred to as a *series parallel identification model* in recognition of the fact that the actual state of the unknown system (rather than that of the identification model) is fed into the identification model, as depicted in Fig.7.6 (a).

The series-parallel identification model of Fig.7.6 (a) should be contrasted against a *parallel identification model* where the $x(n)$ applied to the neural network *I* is replaced with $\hat{x}(n)$; the $\hat{x}(n)$ is derived from the networks own output $\hat{x}(n+1)$ by passing it through a unit delay $z^{-1}I$. The practical benefit of this alternative model of training is that the neural network model is operated in exactly the same way as the unknown system, that is, the way in which the model will be used after the training is completed. It is therefore likely that the model developed via the parallel training mode may exhibit autonomous behaviour that is superior to the autonomous behaviour of the network model developed via the series-parallel training mode. The disadvantage of the parallel training mode, however, is that it may take longer than the series-parallel training mode.

7.6.2 Input-Output Model

Suppose that the unknown plant is only accessible through its output. To simplify the presentation, let the system be of a SISO type. Let $y(n)$ denote the output of the system due to the input $u(n)$ for varying discrete-time n. Then, choosing to work with the NARX model, the identification model takes the form:

$$\hat{y}(n+1) = \varphi(y(n),\dots, y(n-q+1), u(n),\dots, u(n-q+1))$$

where q is the order of the unknown system. At time $n+1$, the y past values of the input and they past values of the outputs are all available. The model output $\hat{y}(n+1)$ represents an estimate of the actual output $y(n + 1)$. The estimate $\hat{y}(n+1)$ is subtracted from $y(n + 1)$ to produce the error signal where $y(n+1)$ plays the role of desired response. The error $e(n+1)$ is used to adjust the synaptic weights of the neural network so as to minimise the error in some statistical sense. The identification model of Fig. 7.7 is of a series-parallel form (i.e., teacher-forcing form) because the actual output of the system (rather than that of the identification model) is fed back to the input of the model.

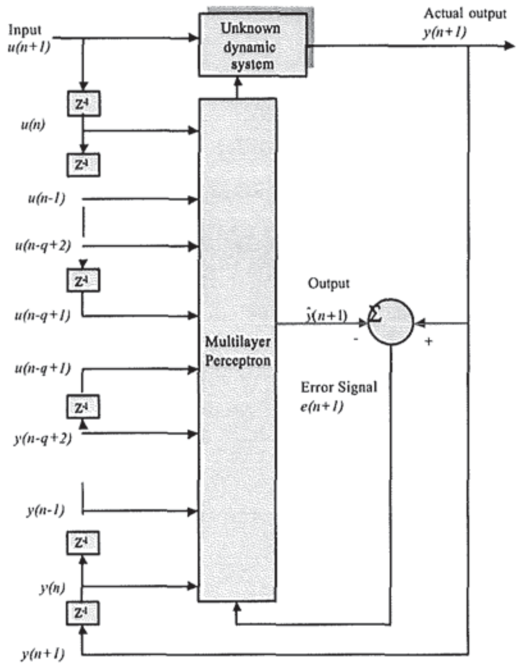


Fig.7.7 NARX solution for the system identification problem.

7.7 Model Reference Adaptive Control

Another important application of recurrent networks is in the design of *feedback control* systems where the states of a plant are coupled nonlinearly with imposed controls [Puskorius and Feldkamp, 1994; Pusk orius *et al.,* 1996]. The design of the system is further complicated by other factors such as the presence of unmeasured and random disturbances, the possibility of a non-unique plant inverse, and the presence of plant states that are unobservable.

A control strategy well suited for the use of neural networks is the model reference adaptive control (MRAC)[4], where the implicit assumption is that the designer is sufficiently familiar with the system under consideration [Narendra and Annaswany, 1989]. Fig. 7.8 shows the block diagram of such a system, where adaptivity is used to account for the fact that the dynamics of the plant are unknown. The controller and the plant form a closed loop feedback system, thereby constituting an externally recurrent network. The plant receives an input $u_c(n)$ from the controller along with an external disturbance $u_d(n)$. Accordingly, the plant evolves in time as a

[4] For early detailed treatment of model reference adaptive control, see the book by Landau (1979)

function of the imposed. Inputs and the plant's own state $x_p(n)$, the output of the plant, denoted by $y_p(n+1)$, is a function of $x_p(n)$. The plant output may also be corrupted by measurement noise.

The controller receives two inputs: (i) an externally specified reference signal $r(n)$, and (ii) $y_p(n)$ representing a one-step delayed version of the plant output $y_p(n+1)$. The controller produces a vector of control signals defined by:

$$u_c(n) = f_1(x_c(n), y_p(n), w)$$

where $x_c(n)$ is the controller's own state and $\mathbf{w}$ is a parameter vector that is available valued function f_1 (.,.,.) defines the input-output behaviour of the controller.

The desired response $d(n+1)$ for the plant is supplied by the output of a stable *reference model*, which is produced in response to the reference $r(n)$. The desired response $\mathbf{d}(n+t)$ is therefore a function of the reference signal $r(n)$ and the reference model's own state $x_r(n)$, as shown by:

$$\mathbf{d}(n+1) = \mathbf{f}_2(\mathbf{x}_r(n), \mathbf{r}(n))$$

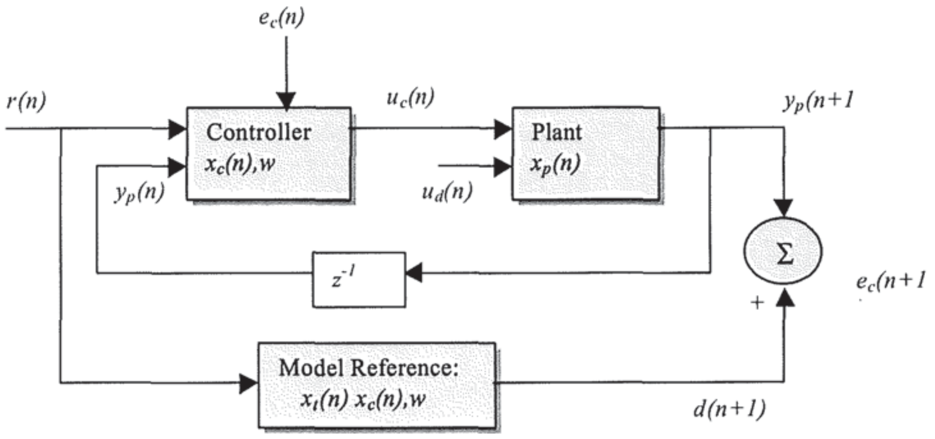


Fig.7.8 Model reference adaptive control using direct control.

The vector-valued function f_t(.,.) defines the input-output behaviour of the reference model.

Let the *output error* (i.e., the error between the plant and model reference outputs) be denoted by:

$$\mathbf{e}_c(n+1) = \mathbf{d}(n+1) - \mathbf{y}_p(n+1)$$

The design goal is to adjust the parameter vector $\mathbf{w}$ of the controller such that the Euclidean norm of the output error $e_c(n)$ is minimised over time n.

The method of control used in the MRAC system of Fig.7.8 is said to be *direct* in the sense that no effort is made to identify the plant parameters, but the parameters of the controller are directly adjusted to improve system performance. Unfortunately, at present, precise methods for adjusting the parameters of the controller based on the output error are not available [Narendra and Parthasarathy, 1990]. This is because the unknown plant lies between the controller and the output error. To overcome this difficulty, we may resort to the use of indirect control, as shown in Fig. 7.8. In this latter method, a two-step procedure is used to train the controller:

1. A model of the plant P, denoted by $\hat{P}$, is obtained to derive estimates of the differential relationships of the plant output with respect to plant input, prior plant outputs, and prior internal states of the plant. The procedure described in the previous section is used to train a neural network to identify the plant; the model $\hat{P}$ so obtained is called an *identification model.*
2. The identification model $\hat{P}$ is used in place of the plant to derive estimates of the dynamic derivatives of the plant output with respect to the adjustable parameter vector of the controller.

In indirect control, the *externally recurrent network* is composed of the controller and input/output representation of the plant via the identification model $\hat{P}$.

The application of a recurrent network to the controller design in the general structure of Fig.7.8 has been demonstrated in a series of example control problems ranging from the well-known cart-pole and bio-reactor benchmark problems to an automotive subsystem, namely engine idle-speed control [Puskorius and Feldkamp, 1994; Puskorius *et al.*, 1996].

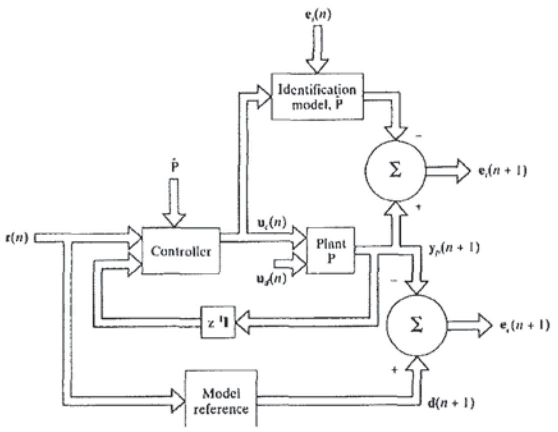


Fig.7.9 MRA control using indirect control via an identification model.

7.8 Summary and Discussion

In this chapter has discussed recurrent networks that involve the use of *global feedback* applied to a static multilayer perceptron. The application of feedback enables neural networks to acquire state representations making them suitable devices for diverse applications in signal processing and control. Four main network architectures belonging to the class of recurrent networks with global feedback are identified:

- Non-linear auto-regressive with exogenous inputs (NARX) networks using feedback from the output layer to the input layer.
- Fully connected recurrent networks with feedback from the hidden layer to the input layer.
- Recurrent multilayer perceptron with more than one hidden layer, using feedback from the output of each computation layer to its own input.

In all of these recurrent networks, the feedback is applied via tapped-delay-line memories.

In theory, a recurrent network with global can learn the underlying dynamics of a *nonstationary* environment and do so by storing the knowledge gained from the training sample in a *fixed* set of weights. The network can *track* the statistical variations of the environment provided that two conditions are satisfied [Haykin, 1999].

- The recurrent network does not suffer from under-fitting or over-fitting.
- The training sample is representative of the non-stationary behaviour of the environment.

The main idea behind the approach described herein is the "generalised recursive neurone," which is a structural generalisation of a recurrent neurone (i.e., neurone with local feedback). By using such a model, supervised learning algorithms such as back-propagation through time and real-time recurrent learning can be extended to deal with structured patterns.

CHAPTER 8: NEURAL NETWORKS APPLICATIONS AND THEIR BENEFITS

8.1 Introduction

The objectives of this chapter is to highlight the potential benefits and application of neural networks for engineering, science and other state of the art technology which has been recently becoming very active area of research.

8.2 Industrial applications

Neural networks applications range from pattern recognition of geophysical features to space applications and some of which are listed below.

Neural networks have been applied in many other fields since the DARPA 1988 report was written. A list of some applications mentioned in the literature is as follows:

Industries	Applications
• Aerospace	High performance aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations, aircraft component fault detectors
• Automotive	Automobile automatic guidance systems, warranty activity analyzers
• Banking	Check and other document readers, credit application evaluators
• Defence	Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification
• Electronics	Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, non-linear modelling.
• Entertainment	Animation, special effects, market forecasting
• Financial	Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction
• Insurance	Policy application evaluation, product optimization

• Manufacturing	Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modelling of chemical process systems
• Medical	Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency room test advisement
• Oil and Gas	Exploration
• Robotics	Trajectory control, forklift robot, manipulator controllers, vision systems
• Speech	Speech recognition, speech compression, vowel classification, text to speech synthesis
• Securities	Market analysis, automatic bond rating, stock trading advisory systems
• Communications	Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems
• Transportation	Truck brake diagnosis systems, vehicle scheduling, routing systems

Aircraft control

A number of aircraft-related control problems have been tackled by using neural network. These include:

(i) Real-time prediction of the unsteady aerodynamics to enhance the aircraft control [Faller and Schreck, 1995].

(ii) 3-dimensionat dynamic reattachment modelling over a broad parameter range [Faller *et al.*, 1995].

(iii) Predicting rotor system component loads during high-speed manoeuvering flight [Haas *et al.,* 1995].

(iv) Amin *et al.,* (1995), applied the neural network controller to control aircraft encountering wind shear on take-off.

(v) Ha (1995) discussed designing a discrete-time lateral-directional control law for a high performance aircraft using neural network.

(vi) Control of aircraft flare and landing has also been reported [Jorgensen and Schley, 1990].

Electric Power control

Adaptive control using neural network has been applied to a variety of electric power control problems. For example:

(a) Maintaining voltage stability for power systems is described in [Jeyasurya, 1994].

(b) Optimal power flow to minimise the cost of power generation is discussed in [Chowdhury, 1992].

(c) To assess the dynamic security of interconnected power can systems using an integrated neural network and modern control theory [Marpaka *et al.*, 1994].

(d) Other design methods for neural network control of voltage and power can be found in [Kojima *et al.*, ,1995; Abdulrahman *et al.* ,1995; Su and Lin , 1995; Auckland *et al.* ,1995].

Robotics control

Current industrial robots are limited in their capabilities. The neural network approach has the potential for adding the ability to learn and adapt and for integrating information from multiple inputs. Most of the robotic applications are in the following two general classes:

I. *Mobile robot control*: This refers to a robot learning to navigate to a designated target. Watanabe *et al.,* (1995) used a hybrid fuzzy-neural network controller for tracking control of a mobile robot driven by two independent wheels. Controlling a 2 degree-or-free mobile robot in a non-stationary environment was described by [Zalama *et al.*, 1995] using an unsupervised neural network.

II. *Trajectory control*: This refers to controlling the robot arm to follow a desired path. In this area, there are several contributors, each using different techniques.

Process control

Process control refers to controlling a complicated industrial or chemical process. Examples of process control using neural network are:

(1) Hydrometallurgical processes control by [Aldrich *et al.,* 1995]. In Aldrich's paper a self-organizing neural network is used to monitor the behaviour of an industrial platinum flotation plant.

(2) Fed-batch fermentation process control is discussed in [Boskovic and Narendra, 1995].

(3) Using a neural network to model and control a pack distillation column is described in [Macmurray and Himmelblau, 1995].

(4) Song *et al.* (1994) applied the neural network controller for gas turbine.

(5) Loh *et al.* (1995) described the control of a pH process.

Temperature control

Industrial temperature control is traditionally done with simple PID controllers. Conventional adaptive controllers such as the Eurotherm self-tuning controller have been successfully applied to temperature control recently. Temperature control using neural network, like many other applications, is still in an experimental state (Ng 1997). Some examples are as follows:

(1) Temperature control of an experimental semi-batch pilot plant reactor equipped with a monofluid heating-

cooling system.

(2) Controlling the temperature of a reactor.

(3) In improving the performance of heating, ventilating and air-conditioning (HVAC) control systems.

Other Applications

Listed as follows are some other known applications of using neural network for control.

(1) Induction motor modelling and control, speed control of a field-oriented induction motor, traffic control, Intelligent traffic management control system, modelling and control of the depth of anaesthesia in the medical field [Rehman *et al.* 1993].

(2) Controlling unstable active magnetic bearing system, Controlling general automaton tasks can be found in [Tolle, 1994].

(3) Spacecraft attitude control [Krishna Kumar, 1994].

(4) Position control in servomechanisms [Lee *et al.* 1994].

8.3 Benefits of Neural Networks

Neural networks offer the following benefits:

- **Ability to tackle new kinds of problems**

  Neural computers are particularly effective at solving problems whose solutions are difficult, if not impossible, to define. This has opened up a new range of applications formerly either difficult or impossible to computerise.

- **Robustness**

  Neural networks tend to be more robust than their conventional counterparts. They have the ability to cope well with incomplete or "fuzzy" data and can deal with previously unspecified or un-encountered situations. Because data and processing are distributed rather than centralised, neural networks can be very tolerant of faults if properly implemented. This contrasts with conventional systems, where the failure of one component usually means the failure of the entire system.

- **Fast processing speed**

  Because they consist of a large number of massively interconnected processing units, all operating in parallel on the same problem, neural networks can potentially operate at considerable speeds. This contrast to the serial, one step at a time processing used in conventional computers.

- **Flexibility and ease of maintenance**

  Neural computers are very flexible in the way in which they are able to adapt their behaviour to new and changing environments. They are also easier to maintain, with some having the ability to learn from experience in order to improve their own performance.

- **Control of industrial processes**

  A neural computer learns to control an industrial process such as brewing by being trained that the correct response to a range of situations, such as temperatures, pressures etc., [DTI]. The network generalises from the cases that it has been trained with so that it can respond to situations that it has not seen before. It may, therefore, be controlling the temperature of a furnace, for example. Such applications have demonstrated a marked increase in accuracy, resulting in cost savings from better control of power and wastage.

- **forecasting movements in financial markets**

  A neural computer is trained to identify the links between conditions and subsequent movements in a particular financial market using historical data. Thereafter, it is able to predict future movements on the basis of current market conditions. The neural network approach has been shown to give more accurate predictions than other methods in this field (DTI)

- **better targeting of mailshots.**

  A neural computer is trained to recognise the links between offers and responses in a mailing list. On the basis of what it has learned, the network is better able to target a specific sub-set of the mailing list for any given new offer. There is a marked increase in the cost-effectiveness of direct mail in such applications.

These following three sample applications have a number of critical factors in common, as well as the potential that computerisation often holds for a high gain from a small increase in performance, that suggest that an approach using neural network should be considered DTI:

1. a complex situation in which the links between inputs and outputs cannot be explicitly specified;
2. an application, or a degree of performance, that cannot be cost-effectively achieved using conventional computing methods;
3. the availability of data to train a neural network.

These applications show how the basic capability of neural computing, to learn their behaviour, has been successfully applied to complex areas such as monitoring and control and prediction. The following list summarises the range of neural network application areas that have often proved difficult to tackle using conventional computing (DTI):

- classification
    - in marketing: consumer spending pattern classification;
    - in defence: radar and sonar image classification;
    - in agriculture & fishing: fruit and catch grading;

- in medicine: ultrasound and electrocardiogram image classification;

- recognition and identification;
    - in general computing and telecommunications: speech, vision and handwriting recognition;
    - in finance: signature verification and bank note verification;
- assessment;
    - in engineering: product inspection;
    - monitoring and control;
    - in defence: target tracking;
    - in security: motion detection, surveillance image analysis and fingerprint matching:

8.4 Conclusions

The applications of neural networks for control community and other field of science and technology certainly look promising; the breadth of interest in this technology have been growing rapidly; however, most of the applications presented in the literature are only realized in simulation as case studies. More research is still needed in neural network theory and its suitability to real practical control applications, which are currently still lacking. Furthermore, majority of neural networks reported in the literature are often trained off-line before being applied to control.

CHAPTER 9: CONCLUSIONS & FUTURE WORK

In this research work, neural network application, in relation to the self-tuning adaptive control of non-linear systems, have been investigated. This started with an overview of the neural networks prospective and their relevance to self-tuning adaptive control.

A comprehensive investigation of neural network topology and various learning algorithms, mainly supervised learning is presented. This thesis is also concerned with the use of efficient algorithms in neural networks for control system application. From this perspective, their main potential lies in the field of non-linear dynamic systems. There exist a number of advantageous properties of neural networks, both static and dynamic, from the viewpoint of non-linear self-tuning adaptive control design and these have been analysed and highlighted in this work.

Application of dynamic networks for non-linear adaptive control has been investigated. The algorithm utilising such a network as a model of the controlled non-linear plant has been proposed. Simulation results have been carried out for specific examples using different methodologies. An extended version of the algorithm has been applied to MIMO control and simulation shows very encouraging results.

9.1 Feedforward Neural Networks

Universal approximation properties of static networks make them a potentially useful tool for non-linear control and identification problems. This property is however not unique to neural networks as there are also other non-linear approximation schemes that exist, e.g., polynomials. A comprehensive comparison of neural networks with such other schemes from this wide subject area, are beyond the scope of this work. However, a number of specific advantageous properties, particularly in the context of adaptive control, unique to neural networks have been emphasised in Chapter 2. Subsequently in Chapter 3, the importance of welknown and up to date control methodologies for the design of non-linear adaptive control are described.

9.2 Recurrent Neural Networks

The main emphasis here is put on applications of recurrent neural networks in non-linear adaptive control. Their use as dynamic models of the controlled non-linear and partly unknown systems, for the case of output feedback, has been advocated in Chapters 4 and 7. Recurrent networks of the Hopfield type have the ability to provide models of non-linear dynamic systems, as described in Chapter 2, and simultaneously appear quite "manageable" allowing analytical treatment.

One of the main theoretical issues of a control system, which has direct implications for system performance, is stability. This is a very complex problem and requires extensive mathematical analysis with much experience especially, in case of non-linear adaptive systems. Hence it is left aside for the future studies. However, here stability strategies have been applied pragmatically. Although some papers have already suggested the use of recurrent networks for identification and control, very little has so far been done in the adaptive context, in terms of using neural network in the stability analysis of control systems.

In Chapter 4 neural network based system identification and self-tuning control for non-linear system algorithms are presented and the effectiveness of the algorithm has been demonstrated by means of simulations examples. These simulation results indicate that it is very promising for control of more complex non-linear plants. The self-tuning adaptive control is traditionally limited to unknown linear systems. By introducing back-propagation neural networks into the self-tuning scheme, it is demonstrated that this new technique has the potential to deal with unknown linearizable non-linear systems.

Chapter 5 describes various fundamental methodologies of adaptive and nonadaptive control system strategies such as GPC and MV. These methods are validated and compared using simulations. The techniques described here are linked directly or indirectly to earlier chapters especially Chapter 4 and Chapter 6

In Chapter 6, neural network enhanced generalised minimum variance control for non-linear system is presented and demonstrated by means of simulations. Here back-propagation neural networks are introduced into the self-tuning scheme; it is demonstrated that this new technique has the potential to deal with unknown linearizable non-linear systems. A neural network enhanced self-tuning controller, which is designed by amalgamating neural network mapping with a generalised minimum variance self-tuning control (GMVSTC) strategy is also examined. Simulation results are presented to illustrate the algorithms described in the chapter.

The main advantages of this non-linear modelling design approach are that the mathematical complexity ordinarily associated with an overall non-linear design can be avoided and computer implementation effort for on-line real-time can be reduced significantly. The main disadvantage is that learning can be slow if the sampling time is too long, however, this can be overcome by using an updating algorithm and / or a fast processor.

A salient point observed during the simulations that this minimum variance of self-tuning adaptive controller might be limited in the real practical environment amongst other possibilities this may be due to noise actuator and / or output constraints. A lesson to be learned is that a self-tuning controller must incorporate a check for unreasonable values of the estimated parameters within its software, so as to avoid blow up.

The potential benefit and applications of neural networks have been presented in Chapter 8 with much focus on control of nonlinear system application.

This thesis also looks at the requirements of neural network simulations from the standpoint of algorithm development and near term application implementations. All the simulation examples carried out in this thesis are simulated using specialised toolbox developed during this research. In the future these algorithm will be continuously updated.

9.3 Existing limitations and further research

The neuro-control analysis presented in Chapter 4 and Chapter 5 is based on the assumption that a choice of the neural model parameters exist which are capable of exactly modelling the plant.

Understanding the use of neural networks with respect to many types of existing model structures and control strategies seem to be not that complex, however it requires very careful understanding of the terminology used by different groups of the neural network and general control communities. Table 9.1 gives some idea of the many theories and methods that are involved with this use of neural networks in identification and control. A new terminology has emerged in the theory of neural networks, such as: feed-forward networks, recurrent networks, supervised learning, unsupervised learning, learning set, test set, generalisation etc. The following links exit between the field of neural networks and control theory.

Neural Networks	**Control Theory**
Feed-forward network	Static non-linear model
Recurrent net	Dynamic non-linear model
Learning	Optimisation
Training set	I/O data for identification
Test set	Fresh data
Generalisation	Cross validation

Table 9.1: Neural network versus existing theory

Applications are expanding because neural networks are good problem solvers, not just in engineering, science and mathematics, but in medicine, business, finance and literature as well (See Chapter 8). Their application to a wide variety of problems in many fields makes them very attractive. Also, faster computers and faster algorithms have made it possible to use neural networks to solve complex industrial problems that formerly required too much computation.

9.4 Direction of Future Research

Although volumes of journals, articles and books on neural networks for control exists, there is still much research needed especially in the area of self-tuning adaptive control of unknown non-linear systems. The following areas still needing investigation before significant industrial applications can be achieved are:

1. The control stability is a very complex problem mathematically, especially the discrete form in adaptive control. Hence the neuro control structure is another ongoing research topic. It is known that the stability of a closed loop system is not implied by the stability of the open loop system. This situation recurs in a more complex form in adaptive neuro control: a well-behaved *off-line* parameter estimation algorithm can become unstable when operating *on-line* via a feedback controller. This could happen when the identification of the actual process contains 'unmodeled high frequency dynamics" and is thus of higher order than the control model. For example, when rapidly varying (high frequency) control signal hits the process or when the plant dynamics are unmodeled resulting in slow response to the output process, etc.
2. Global convergence and stability. There is a need for theoretical analysis of global convergence and stability in using NNs for control. Global convergence and stability results are important in gaining the confidence in industries. One possibility for achieving this is to use the hybrid strategy discussed in Chapter 3. In the hybrid strategy, the conventional controller such as feedback controller based on some knowledge of the plant, could be designed, to ensure global stability of the control systems. Using other conventional controllers in the hybrid strategy such as fuzzy logic, expert control and knowledge based controllers should also be investigated.
3. Network topology. Current work assumes that the number of units in the neural network controller is chosen correctly for the plant to be controlled. However, in practical applications, the necessary prior knowledge may not be available. Hence learning algorithms which automatically vary the network topology, such as the number of units required, would be desirable. These may also include determining the number of relevant past plant inputs and outputs needed when the order of the plant is unknown. Alternatively, theoretical results have to be established to justify the use of a sufficient number of units for a NN to represent a plant properly.
4. Current work also assumes that the order of the plant is known and all plant states are measurable. In situation where this assumption is not valid, a dynamic unknown plant may have to be controlled using feed-forward recurrent neural network. FRNN is not popularly used because of high computational cost. However, initial simulations have shown that it is feasible to use fixed recurrent weights to reduce the computational cost of FRNN. Nevertheless, more investigations are needed to justify the use of FRNN with fixed recurrent weights. Single layer RNN should also be investigated.
5. Supervisory control. This work has demonstrated the use of NNs for direct control of unknown plant. However, about 90% of industrial applications are still using conventional controllers such as PID controller [Seborg, 1994]. Human supervisors are often employed for higher level control and decision making of these conventional controllers. Since neural networks have the capability to mimic a human expert, the potential for them to be used in place of human supervision for certain tasks are worth investigating.

9.5 Other points to note

- Software development will continue to keep pace with the hardware development with a finite lag time.

Commercial software available today allows for small network development at low costs. The commercial marketplace and individual needs will drive further development in both the commercial or private arenas. One of the reasons for the current advancements in neural networks has been the availability of affordable, easy-to-use computing facilities. Current applications have exploited these facilities. When the capacity of these facilities is compared to possible applications in signal processing, robotics, speech and vision the facilities are deficient.

- Simulations will play an important role in the development of neural network algorithms and applications. In fact, most short-term applications will be implemented as a simulation.
- Neural network algorithms are presented in numerous mathematical forms. One of the popular forms is as a set of coupled differential equations. Current hardware accelerators used for neural network simulations do not easily allow (if at all) the use of such equations. Thus, *it is recommended in the DARPA researcher scientist that the development of hardware and software simulators using the differential equation descriptions be encouraged* [DARPA1988].
- Simulations are primarily used to understand the dynamics of a particular network and for modest implementations. As the need for high-speed, low-cost, low-power and small size implementations increases, so will the need to understand the characteristics of the devices used in these advanced implementations. It will be through simulations that the characteristics of implementations are studied and understood. Future simulation requirements must, therefore, account for such simulations of implementations. This type of simulation is more demanding than those chiefly to understand network dynamics or for small-scale implementations because the later type of simulations much encompasses the dynamics of both the network and the devices [DARPA].
- The availability of high-end simulation tools to the neural network research community is critical for the development of near-term applications. Many researchers throughout the study noted that the lack of inexpensive and easily accessible simulation facilities inhibit their research. Current hardware accelerators have brought a significant increase in computational power to researchers [DARPA 88].

9.6 What Neural Network Technology can offer?

On the basis of comparison between neural networks and other information processing technologies, the following conclusions appear to be a fair assessment of neural network technology:

- Neural networks offer significant potential benefits for information processing, such as knowledge acquisition through learning, fast processing speeds, robustness to implementation defects, and compact processors.
- The prime candidates for early neural network applications are expected to be in the areas of pattern classification, simple computational maps for robotic control systems, early vision, signal processing, and speech recognition.
- Neural network technology is not mature enough at present for widespread practical applications, since

computer simulations presently are the primary method of implementing neural networks while hardware implementations remain in the experimental stage.

- The first hardware implementations will undoubtedly be functional modules for inclusion in systems using conventional technologies.
- Realising the potential benefits of neural network technology will require basic research to advance the technology on several fronts, including:
  - Theory, including representations, efficient learning algorithms, stability (See section 9.4)
  - Modular architectures, overall system control; and
  - Implementation techniques for silicon and optics.
- The generic application areas to be pursued should be those where success from the unique neural network approach is likely and would have an important impact. This comparison study indicates that these areas are pattern classification, early vision, speech recognition, signal processing, robotic control and so on.
- The program should establish a methodology that allows measurement of progress toward goals by providing specified performance criteria, benchmark problems and databases, and review of unsuccessful as well as successful projects, to make best use of experience from the program.
- A neural network program should assure good coupling to other branches of information processing, neuroscience, and cognitive science to take advantage of conceptual breakthroughs in the difficult application areas such as vision and speech and in the understanding of particular simple biological neural networks, notably invertebrates.

9.7 Important Issues raised within this thesis are:

- Determination of combining models as well as the combination process.
- The excitement of studying neural nets in regard to their potential in applications for which solutions have not been found through conventional computing.
- Neural network alogrithms show immediate advantages over existing algorithms for certain application areas.
- Successful neural network applications, where correct and knowledgeable representations are crucial.
- Algorithmic and symbolic processing elements which will be combined in future systems.
- The cases where numerous researchers are giving their attention to the new science but applying neural networks to simplified problems to test their ideas, rather than solving problems on a realistic scale.

In this thesis neural models obtained using identification are intended to become a part of the control structure together with self-tuning adaptive controller. According to the simulation results presented in chapter 4, 5 and 6, this shows that by using the BP NNs into identification and self-tuning control scheme has the potential to deal with unknown feedback non-linear system. It is still under investigation to improve the quality of control, efficiency of speed and stability of the control system. The predictive control strategy applied here has many advantages. It is easy to tune and provides a good controller performance. It is also able to account for constraints on input output etc.

9.8 Final remark

In this research, significant contribution is dedicated to the algorithms and the network topology. Controlling of *unknown* complex systems is a very difficult problem, especially when the *black-box* systems are highly non-linear. The simulation results have shown that neural networks can overcome this problems and hopes to have contributed to setting the foundation for the further development of neural network self-tuning adaptive neural network based control system technology.

System Identification and self-tuning controller of non-linear plant

- Without controller the plant produces wild oscillatory when excitation signal is applied.
- With PI controller, slight reductions of oscillation are observed when compare to the case without controller. PI controller cannot handle non-linear plants.
- With GPC, oscillations are reduced dramatically and plant output approximates to reference signal. Quite robust.
- With NN controller
  - better performance is obtained when compared to GPC and PI
  - due to learning capabilities plant output closely matched to the reference signal.
  - after certain time, however, a sudden burst occurred and neural network stopped learning. It needs to be retrained.
  - possible improvement - adjust parameters of the neural network controller

Generalised Minimum Variance Self-tuning Control

The network was trained three times in each sample period. The controlled system outputs are shown in Fig.6.8. They show a good tracking of reference signal, which imply that the respectable decoupling control is achievable. The output values from the non-linear submodel and the BP network and the estimated linear submodel parameters are shown in Fig.6.9 and Fig.6.10 respectively whilst the controller output is shown in Fig.6.11.

Remarks from simulation observation

- Successful parameter estimation can be performed with the aid of a forgetting factor of less than unity, but only if an appropriately changing input is present.

- In practice we have to fix the size of the neural network before it is used in the adaptive control system. Thus, the modelling error is determined.
- A lesson to be learned is that a self-tuning controller must incorporate a check for unreasonable values of the estimated parameters within its software, so as to avoid blow up.
- This minimum variance of self-tuning adaptive controller may be limited in the real practical environment due to noise actuator and output constraints etc.
- By introducing BPNNs into the self-tuning control scheme, it is demonstrated that the new control method has the potential to deal with unknown feedback linearisable non-linear systems.
- The predictive control strategy applied here has many advantages, it is easy to tune, it is flexible and provides a good controller performance.
- Simulation results indicate that the BP neural learning has the capability to learn arbitrary non-linearities and show great potential for self-tuning tracking problems in control applications.
- The neuro based controller is superior than GPC and PI controllers in handling non-linear problems.
- Different types of excitation signals can be applied and performance can be observed in a real time environment.
- There are currently no commercial software packages available exclusively for system identification and self-tuning adaptive control system design, however, general-purpose neural networks programs are available.
- Programs are easily expandable to incorporate and test new methods. Programs are easy to update without going through program listing.
- The tools have been developed for the MATLAB environment for several reasons: MATLAB is a very versatile numerical software package that runs on most hardware platforms and it has an extensive interactive environment for data visualisation.

REFERENCES

[1] Abdelaziz A.Y., M.R.Irving, M.M.Mansour, A.M.El-Arabaty and A.I.Nosseir (1996). Adaptive detection of generator out-of-step conditions in power systems using an artificial neural network. *UKACC Int. Conf. on CONTROL'96. Conf. Publication* No: 427 © IEE, p166-171.

[2] Abdulaziz, A. and M.Farsi (1993). Dynamic modelling and control for a class of non-linear systems using neural nets. In '*IEEE Int. Sysmp. Industrial Electronics*'. IEEE, New York. Budapest, Hungary p543-548.

[3] Agarwal, M. (1994). A systematic classification of neural network based control. In 'IEEE Conference on Control Applications. Proceeding'. IEEE. New York. P 149-159.

[4] Aggarawal Raj and Yonghua Song (1998). Tutorial: Artificial Neural Networks in Power Systems. *Part 2* Types of artificial neural networks. *IEE Power Engineering Journal*. Vol. 12 (1) p41-47.

[5] Aggarawal Raj and Yonghua Song (1998). Tutorial: Artificial Neural Networks in Power Systems. *Part 3*: Examples of applications in Power systems. *IEE Power Engineering Journal*. Vol. 12 (6) p279-28.

[6] Albertini F.and E.D.Sontag (1992). For neural networks, function determines form. Neural Networks. Proc. *IEEE Conf. Decision and Control*, Tucson, p26-31.

[7] Albertos P., R. Strietzel and N.Mort (1997). Control Engineering Solutions a practical approach. Control Engineering Series 54: The Institution of Electrical Engineers, London, UK

[8] Aldeen M. and J.F.Marsh. Stabilisation of power systems by a reduced dynamic output controller. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p158-161.

[9] Aldrich, C., D.W.Moolman and J.S.J.Vandeventer (1995). 'Monitoring and Control of hydrometallurgical processes with self-organising and adaptive neural net systems'. *Computers and Chemical Engineering* Vol.19(SS), 803-808.

[10] Alexander G.Parlos, Omar T.Rais and Amir F.Atiya (1999). Multi-Step-Ahead Prediction Using Dynamic Recurrent Neural Networks. *IEEE, Int.Joint Conf. on Neural networks (IJCN)* Vol. 1, p349-352.

[11] Amari S.Z (1990). Mathematical foundations of neuro-computing. IEE Proc. 1990 Vol.78(9) pp1443-1463,

[12] Amin S.M., E.Y.Rodin (1997). Application of dynamic neural networks to approximation and control of non-linear systems. *Proceeding of the 1997 American Control Conference* (ACC), Vol. 1 p222-226.

[13] Anderson J.A and E.Rosenfeld. *Neurocomputing: Foundation of Research*. MIT Press, Cambridge, MA, 1988.

[14] Anderson, J.A., J.W.Silverstein, S.A.Tsoi (1995). " Single net indirect learning architecture'. *IEEE Trans. On Neural Network* 5(6), 1003-1005. *

[15] Andrew R. Barron (1993). Universal Approximation Bounds for Superposition of a Sigmoidal Function. *IEEE Tras. on Information Theory*, Vol. 39 (3), p930-945.

[16] Andrew G.Barto, Richard S.Sutton and Charles W.Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man and Cybernetics SMC*-13: p834-846. (Also found in Neurocomputing (2nd printing 1988).

[17] Arahal ,M.R. et al. (1998). Neural identification applied to predictive control of solar plant: *Control Eng.*

Pract. (UK) Vol. 6(3) p333-344.

[18] Arelhi R., J.Wilkie and M.A.Johnson. On stable LQG Controllers and Cost functional values. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p270-275

[19] Åström Karl Johan and Björn Wittenmark: 2nd Ed. Computer Controlled Systems Theory and Design. © 1990, 1984 by Prentice-Hall.

[20] Åström Karl Johan and Björn Wittenmark: 3rd Ed. Computer Controlled Systems Theory and Design. © 1998 Prentice-Hall.

[21] Åström K.J., H. Panagopoulos and T.Hagglund. (1998). Design of PI Controllers based on Non-convex Optimization. *Automatica* Vol. 34 (5), p 585-601.

[22] Åström Karl Johan and Björn Wittenmark (1989). Adaptive Control. Addison- Wesley Publishing Company.

[23] Åström Karl Johan and Tore Hägglund (1988). Automatic Tuning of PID controllers. © Instrument Society of America 1988 (ISA).

[24] Athans M. (Guest Editor)(1971). Special Issue on the LQG Estimation and Control Problem. ©*IEEE Trans.on Automatic Control*. AC-16 (6), p527-527.

[25] Athans M. (1971). "The Role and Use of the Stochastic Linear Quadratic-Gaussian Problem in Control System Design. © IEEE Trans. Automatic Control, AC-16 (6) p529-552.

[26] Atherton D.P.and A.F.Boz (1998). Using standard forms for controller design. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 © IEE, p1066-1071.

[27] Ayoma, A., F.J.Doyle and V.Venkatasubramanian (1996). 'Control affine neural network approach for non-minimum phase nonlinear process control'. *Journal of Process Control* Vol.6 (1), 17-26.

[28] Balakrishnan, S.N. and R.D.Weil (1996). 'Neurocontrol – a literatrue survey'. *Mathematical and Computer Modelling* Vol. 23 (1-2), p101-117.*

[29] Baldi P. (1995). Gradient descent learning algorithm overview: A general dynamical systems perspective. *IEEE Trans. on Neural Networks* Vol.6 (1), p182-195.

[30] Barto, A.G., R.S.Sutton and C.W. Anderson (1983). 'Neuronlike adaptive elements that can be solve difficult learning control problems'. IEEE Trans. on System Man and Cybernetics 13(5), 834-846. *

[31] Bahram, Shahian, Michael, Hassul. Control System Design using Matlab. © Prentice-Hall, Inc. A Simon & Schuster Company Englewood Cliffs, New Jersey, 1993.

[32] Barren, A. (1993). Universal approximation bounds for superpositions of a sigmoidal systems perspecitve. *IEEE Trans.on Neural Networks* Vol. 6 (1), p182-945.

[33] Bennett B.S.(1995). Simulation Fundamentals. 1st Ed. Parentice hall International (UK).

[34] Bellman R. and R.Kalaba. Dynamic Programming and Feedback Control. *Automatic and Remote Control Proceeding of the 1st International Congress of the I.F.A.C.*Moscow, 1960.

[35] Bittani S. and L.Piroddi (1993). Minimum Variance Control of a class of non-linear plants with Neural networks. In: *3rd Int.Conf.on Artificial Neural networks*, Brighton, UK.

[36] Blachuta M.J. A unified state-space approach to LQG and GPC controller. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p66-71.

[37] Blaschke, F (1972). The principle of field orientation applied to the new transvector closed-loop control system for rotating field machines. *Siemens Review* Vol. 39, p217-220.

[38] Billings S.A., J.O.Gray and D.H.Owens.(1984). Non-linear System Design. © IEE Control Series 25, Peter Peregrinus Ltd.

[39] Bilings S.A.and Q.M.Zhu. (1995). Model validation tests for multivariable non-linear models including neural networks. *Int.J.Control*, Vol. 62 (4) p749-766.

[40] Bikfalvi P. and I. Szabo.(1997) Classic Controller. In Control Engineering Solutions a practical approach. IEE series 54. Ch.3 p 43-59.

[41] Bose, J,A. and Liang (1996). Neural Network Fundamentals with Graphs, algorithms, and Applications. McGraw-Hill, Inc.

[42] Boskovic, J.D. and K.S.Narendra (1995). ' Comparison of linear, nonlinear and neural network based adaptive controller for a class of fed batch fermentation processes'. *Automatica* Vol. 31 (6), 817-840.

[43] Brown, M. and C.J.Harris (1994). Nerofuzzy Adaptive Modelling and Control Prentice Hall International.

[44] Brown, R.H., Ruchti, T.L. and Feng, X., (1992) " Artificial neural network identification of partially known dynamic non-linear systems. *Procs. of the 32nd IEEE Conference on Decision and Control,* Vol. 4, p. 3694-9, 1992.

[45] Brown Lyndon J. and Sean P.Meyn. (1993). Prediction and Adaptation of PID Controller Design. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 2 of 4 p1575-1580.

[46] Bryon R.Maner, Frncis J.Doyle III, Babatude A. Ogunnaike and Ronal K.Pearson (1996). Non-linear Model Predictive Control of Simulated Multivariable Polymerization Reactor Using Second-order Volterra Models. *Automatica*, Vol. 32, (9) p1285-1301.

[47] Cabrera J.B.D.and K.S.Narendra (1999). Issues in the application of Neural Networks for Tracking Based on Inverse Control. IEEE Trans. on Automatic Control Vol. 44 (11) p 2007-27.

[48] Chapman C.B., D.F.Cooper, M.J.Page (1987). Management for Engineers Edited. John Wiley & Sons Ltd. U.K.

[49] Charles L. Phillips Royce D. Harrbor (1996). Feedback Control Systems. 3rd Edition by Prentice-Hall, Inc. A Simon & Chuster Company Englewood Cliffs, New Jersey.

[50] Charles L.Phillips and H.Troy Nagle (1995). Digital Control System Analysis and Design. 3rd Ed. Prentice-Hall.

[51] Chang Timothy N. & Nirwan Ansari (1993). A Learning Controller for the Regulation and Stabilization of Flexible Structures. *Proc.of the 32nd Conf.on Decision and Control San Antonio*, Vol. 2 of 4 p1268-1273.

[52] Chan H.L., A.B.Rad (2000), Real-time flow control using neural networks. ISA Transactions Vol. 39, p93-101.

[53] Chengyn Gan and Kourosh Danai (2000). Model-Based Recurrent Neural Network for Modelling Non-linear Dynamic Systems. IEEE Trans. on systems, Man. And Cybernetics-part B, Vol. 30 (2), p344-351.

[54] Chen Fu-Chung (1990). Back-propagation neural networks for Non-linear self-tuning Adaptive Control. *IEEE Control Systems Magazine*. p44-48.

[55] Chen Fu-Chuang (1991). A Dead-Zone Approach in Non-linear Adaptive Control Using Neural Networks.

Proceedings of the 30th Conference on Decision and Control Brighton, England 1991, p156-161.

[56] Chen Fu-Chuang and Hassan K.Khalil (1991). Adaptive Control of Non-linear Systems Using Neural Networks – A Dead-Zone Approach. Proceedings of the 1991 American Control Conference p 667-672.

[57] Chen, Fu.Chuang "Back-propagation Neural Networks for Non-linear Self-tuning Adaptive Control" IEEE Control System Magazine, Special Issue on Neural Networks for Control Systems, April 1990.

[58] Chen, S., Billings, S.A. and Grant, P.M., "Non-linear system Identification using neural networks, " *Int.Journal of Control,* Vol. 51 (6) p. 1191-1214, 1990.

[59] Chen, S., Billings, S.A., C.F.N.Cowen and Grant, P.M. (1990), Practical Identification of NARMAX models using Radial Basis functions, " *Int.Journal of Control,* Vol. 52 (6) p. 1327-1350.

[60] Chen, V.C. and Y.H.Pao (1989). Learning control using neural networks. In: 'Proceedings International Conference Robotics and Automation'. Vol.3 p.1448-1453.

[61] Chichochi, A., and Unbehauen, R. Neural networks for optimisation and signal processing. John Wiley, Chichester, UK, 1993.

[62] Chisci L., A.Garulli and G.Zappa. (1993). Fast Algorithms for LQ Control Design of Time-Invariant Systems. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 1 of 4 p116-1121.

[63] Chowdhury, B.H. (1992). 'Toward the concept of integrated security – optimal dispatch under static and dynamic constraints'. *Electric Power Systems Research* Vol.25 (3), 213-225.

[64] Chipperfield A. J. and P J Fleming. (1993). MATLAB® toolboxes and applications for control. Peter Peregrinus Ltd. IEE controls Engineering Series 48, Peter Peregrinus Ltd., on behalf of the Institution of Electrical Engineers, London, UK.

[65] Chui C. K., G. Chen. (1987). Kalman Filtering with Real-time Application. Springer-Verlag Belin Heidelberg. Druckhaus Beltz, 6944 Hemsbach/Bergstr.

[66] Chiang Richard Y., Michael G.Safonov. (1988-92). Robust Control Toolbox MATLAB® User's Guide by the Mathworks®.

[67] Cooke M.J., G.L.Lebby. An optimal design method for multiayer feed-forward networks. Proceeding of Thirtieth (30th) Southeastern Symposium on system theory (1998) p507-11.

[68] Craddock R. J. and K.Warwick (1998). Dealing with complexity: An overview of the neural network approach. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p705-708.

[69] Craddock R. J. (1997). Multilayered radial basis function networks and the application of state space control theory to feed-forward neural networks. Thesis (PhD) University of Reading, Dep. of Cybernetics.

[70] Clarke D.W (1988). Application of Generalised Predictive Control (GPC). In IFAC Adaptive Control of Chemical Process edited by M.Kumel, p1-8.

[71] Clarke D.W (1990). Application of Generalized Predictive Control to Industrial Processes. *IEEE, Control Systems Magazine*. Vol.8 (2) p49-55.

[72] Cutler C.R. and B.C.Ramaker. Dynamic matrix control- a computer control algorithm. In *Joint Automatic Control Conference,* paper WP5-B, San Francisco, CA, 1980.

[73] Czeslaw Dacka. On the Controllability of Non-linear Systems with Time-Variable Delays. *IEEE Trans.on Automatic Control.*(1981) Vol. AC-26, (4), p956-959.

[74] Edwin K.P.Chong, Stefen Hui and Stanislaw H.Zak (1999). An Analysis of a class of Neural networks for Solving Linear Programming Problems. IEEE Trans. on Automatic Control Vol. 44 (11) p1995-2006.

[75] DARPA Neural Network Study. October 1987 - February 1988. AFCEA International Press, 1988.

[76] David E.Rumelhart. and McClelland, J.L. (Eds.) (1986). Parallel Distributed Processing: explorations in the microstructure of cognition. Vol. 1. London: MIT press. *

[77] David E.Rumelhart, Geoffrey E.Hinton and Ronald J.Williams (1986). Learning representations by back-propagating errors. Nature 323: 533-536. (In Neural Computing: Foundation of Research edited by James A.Anderson and Edward Rosenfeld 4th printing (1988,89).

[78] David E.Rumelhart, Geoffrey E.Hinton and Ronald J.Williams (1986). Learning internal representations by error back-propagation. Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1, D.E.Rumelhar and J.L.McClelland (Eds.), Cambridge, MA: MIT Press, p 318-362. (In Neural Computing: Foundation of Research edited by James A. Anderson and Edward Rosenfeld 4h printing (1988,89).

[79] David G.Luenberger. (1971). An Introduction to Observers. © *IEEE Trans. on Automatic Control*, Vol. AC-16, (6) p596-602.

[80] Davis M.H.A.and R.B.Vinter. Stochastic modelling and control. ©1985.

[81] Daniel Tabak Benjamin C.Kuo. (1971). Optimal Control by Mathematical Programming. Prentice-Hall, Inc., Englewood Cliffs.

[82] Damle, R., R. Lashlee, V.Rao and F.Ken (1994). 'Identification and robust control of smart structures using artificial neural networks'. *Smart materials and Structures* 3 (1), p35-46.

[83] Delgado A., C.Kambhampati and K.Warwick (1996). Stable linearization using multilayer neural networks. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p194-198.

[84] Delgado A., C.Kambhampati and K.Warwick (1995). Dynamic structure neural networks for system identification and control. *IEE Proc. on Control Theory and Applications.* Vol.142, p307-314.

[85] Doyle J.C. (1978). Guaranteed margins for LQG regulators. *IEEE Trans.on Automatic Control*, AC-23, 756-7.

[86] Doyle J.C. and Stein G. (1979). Robustness with observers. *IEEE Trans.on Automatic Control*, AC-24, p607-11.

[87] Doyle J.C. and Stein G. (1981). Multivariable feedback design: Concepts for a classical/modern synthesis. *IEEE Trans.on Automatic Control*, AC-26, p 4 -16.

[88] DTI (1993). Neural Computing *Learning Solutions.* Directory of Neural Computing Suppliers, Products and Sources of Information pack. © Crown copyright 1993.

[89] L.Dugard and J.M.Dion (1985). Direct adaptive control for linear multivariable systems. *Int.J.Control,* Vol.42, p1251-1281.

[90] Duwaish H.AL-, M.Nazmul Karim and V.Chandrasekar. (1997) Hammerstein model identification by multilayer feedforward neural networks. *Int. Journal of Systems Science*, Vol. 28 (1), p49-54.

[91] Duncan, T.E., L.Guo, B.Pasik-Duncan (1999). Adaptive Continuous – Time Linear Quadratic Gaussian Control. *IEEE Trans. on Automatic Control,* Vol. 44 (9), 1999 p1653-1662.

[92] Dutton K. and William A.Barraclough. (1996). A progressive introduction of computer assistance into the teaching of advanced topics in control engineering. © IEE, *Engineering Science and Education Journal* No.5 (1) p32-40.

[93] Dutton K, Steve Thompson and Bill Barraclough. The Art of Control Engineering. © 1996 Addison Wesley Longman.

[94] Dwinnel. W. (1998). Modelling Methodology 3 Algorithm selection. *PC AI (USA).* Vol. 12, (2), p32-35.

[95] Eykhoff, P., *Trends and Progress in System Identification,* Oxford, England: Pergamon, 1981.

[96] Erdal C. A sensitivity measure for an electronic, Proportional-Integral (PI) controller and calculating optimum parameter tolerances. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p275-277.

[97] Esref Eskinat and Stanley H.Johnson (1991). Use of Hammerstein Models in Identification of Nonlinear systems. Use of Hammerstein Models in Identification of Nonlinear systems. *AIChE Journal* (1991), Vol. 37 (2) p255-267.

[98] Faller, W.E., S.J.Schreck (1995). 'Real-time prediction of unsteady aerodynamics application for aircraft control and maneuverability enhancement'. *IEEE Transactions on Neural Networks* Vol. 6(6), 1461-1468.

[99] Faller, W.E., S.J.Schreck and H.E.Helin (1995). 'Real time model of 3 dimensional dynamic reattachment using neural networks'. *Journal of Aircraft* Vol.32 (6), 1177-1182.

[100] Fang Y. and T. Kincaid, (1996a). Stability analysis of dynamical neural networks. *IEEE Trans. on Neural Networks* Vol.7, p996-1006.

[101] Fang Y. and T. Kincaid, (1996b). Sufficient conditions for stability of a class of neural circuits. In: *Proc. Of the 13th IFAC World Congress, San Francisco.* Vol. F. P169-174.

[102] Fletcher, R. (1987). *Practical Methods of Optimisation.* Chichester: Willey

[103] Forti M. and A. Tesi (1994). Conditions for global stability of some classes of nonsymmetric neural networks. In: *Proc. 33rd Conference on Decision and Control, Lake Buena, Vista.* Pp. 2488-2493.

[104] Forti M., S.Manetti and M.Marini (1994). Necessary and sufficient condition for absolute stability of neural networks. *IEEE Trans. on Circuits and System –I: Fundamental Theory and Applications.* Vol. 41, p491-494.

[105] Fortescue T.R, L.S.Kershenbaum and B.E.Ydstie. Implementation of self-tuning regulators with variable forgetting factors. *Automatica,* Vol.**17,** 831-835, 1981.

[106] Francis H.I.Chang and Rein Luus. A noniterative Method for Identification Using Hammerstein Model. *IEEE Trans.on Automatic Control,* (1971) p464 - 468.

[107] Friedman, J. and W.Stuetzel (1981). Projection pursuit regression. *Journal of the American Statistical Association* Vol.76, p817-823.

[108] Fu King-Sun (1970). Learning Control Systems-Review and Outlook. IEEE Trans. on Automatic Control, p210-219.

[109] Fu, L.M (1994). Neural networks in computer Inteligence. London: McGraw-Hill.

[110] Funahashi, K.I. (1989). On the approximation realization of continuous mapping by neural networks. *Neural Networks,* Vol.2, p183-191. ***

[111] Funahashi, K.I. and Y.Nakamura (1993). Approximation of dynamical systems by continuous time recurrent networks. *Neural Networks,* Vol.6, p801-806. ***

[112] Freeman J.A. and Skapura. D.M. (1992). Neural Networks: Algorithms. Applications, and Programming Techniques (Reading, Massachusetts. U.S.A.: Addison-Wesley)

[113] Franois E.Cellier. (1991). Continuos System modelling. Spring-verlag Newyork, Inc.

[114] Furuta K., M.Wongaisuwan, and H.Werner. (1993). Dynamic Compensator Design for Discrete-Time LQG problem Using Markov Parameters. *Proc.of the 32nd Conf.on Decision and Control San Antonio*, Vol. 1 of 4 p96-101.

[115] Gambier A. and H.Unbehauen. (1993). A State-Space Generalized Model-Based Predictive Control for Linear Multivariable Systems and its Interrelation with the Receding Horizon LQG- Control. *Proc.of the 32nd Conf.on Decision and Control San Antonio.* Vol. 1 of 4 p817-822.

[116] Garces F., K.Warwick, and R. Craddock. Multiple PID mapping using neural networks in a MIMO steam generator system. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p503-508.

[117] Gaston F.M.F., D.W.Brown and J.Kadlec. A parallel predictive controller. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p1081-1086.

[118] Gawthrop P. & Eric Ronco. Local Model Networks and Self-Tuning Predictive Control. Report: CSC-96001. April 24, 1996. (Also see www.eng.gla.ac.uk/peterg/)

[119] Gibson and G.H.Lee. (1993). Least Squares Estimation of Linear Systems in the Presence of Noise. *Proc.of the 32nd Conf.on Decision and Control San Antonio.* Vol. 2 of 4 p1570-1574.

[120] Garcia C.E and Morari (1982). Internal model control - I. A unifying review and some new results.. *Ind.Eng.Chem. Process Des. Dev.,* Vol. 21. p308-322.

[121] Girosi, F. and T.Poggio (1990). Networks and the best approximation property. *Biological Cybernetics* 63, p169-176.

[122] Goodwin G.C. and K.S.Sin. *Adaptive Filtering, Prediction and Control.* Prentice Hall, Englewood Cliffs, NJ, 1984.

[123] Graham C.Goodwin and Kwaisang Sin. Adaptive Control of Non-minimum Phase Systems. *IEEE Trans.on Automatic Control*, (1981) Vol. AC-26, No.2, p479-483.

[124] Grace Andrew, Alan J.Laub, John N.Little and Clay M.Thompson. Control System Toolbox MATLAB User's Guide ©1986-92 by The Mathworks®.

[125] Grzegorz J. Kulawski, Mietek A.Brdys (2000). Stable adaptive control with recurrent networks. . *Automatica* Vol. 36, p 5-22.

[126] Greblicki, W. and Pawlak. M. 1986. Identification of discrete Hammerstein systems using Kernel regression estimates. *IEEE Trans.on Automatic Control*, AC31, 1, 74-77:

[127] Greblicki, W. and Pawlak. M. 1989. Identification of discrete Hammerstein systems using Kernel regression estimates. *IEEE Trans.on Automatic Control*, AC35, 1, 409-418:

[128] Grimble M. J., (1987). An introduction to Kalman filters. In Ch 9 p 131-148. O'Reilly John. Multivariable control of industrial applications. IEE Control engineering series 32.

[129] ___________ (1999). Stochastic Control of Discrete Systems: A Separation Principle for Wiener and Polynomial Systems. *IEEE Trans. on Automatic Control,* Vol. 44 (11), p2125-2130.

[130] Gu X.Y., Wang and M.M.Liu. Multivariable generalised predictive adaptive control. In *Proceedings of the* 1991 *American Control Conference.* p1746-1751, Boston, MA, 1991.

[131] Guoping P.Liu, Visakan Kadirkamanathan and Steve A.Billings (1998). On-line identification of non-linear systems using volterra polynomial basis function neural networks. Neural Networks 11, p1645-1657.

[132] Haley Pam, Don Soloway and Brian Gold (1999). Real-time Adaptive Control Using Neural Generalized Predictive Control. Proceedings of the American Control Conference, San Diego, California. p4278-4282.

[133] Hai-Long Pei, Qi-Jie Zhou. Approximate non-linear system linearization with neural networks. *Proceeding of the 1997 American Control Conference* (ACC) Vol. 1, p821-2.

[134] Hang C.C (1998). Adaptive output feedback for general non-linear systems using multi-layer neural networks. *Proceeding of the 1998 American Control Conference* (ACC) (1998) IEEE Vol.1, p 520-524.

[135] Ha, C.M (1995). 'Neural networks approach to AIAA aircraft control design challenge'. *Journal of Guidance, Control and Dynamics* Vol.18 (4), 731-739.

[136] Haas, D.J.,J.Miland and L.Fitter (1995). 'Prediction of helicopter component loads using neural networks'. *Journal of the American Helicopter Society* Vol. 40 (1), 72-82.

[137] Harris C.J and S.A.Billings (Editors): Self-tuning and Adaptive Control: Theory and Applications. IEE Control Engineering Series 15. © 9181: Peter Peregrinus Ltd.

[138] Haykin Simon (1994). *Neural Networks. A Comprehensive Foundation.* Prentice-Hall, Inc.

[139] Haykin Simon (1999). *Neural Networks (2nd Ed.). A Comprehensive Foundation.* Prentice-Hall.

[140] Herbert Werner. Generalized Sampled-Data Hold Functions for Robust Multivariable Tracking and Disturbance Rejection. 36th IEEE Conferences on Decision and Control. © 1997 by the Institute of Electrical and Electronic Engineers, Inc. Vol. 3 p-2055-2060.

[141] Henrici P., Elements of Numerical Analysis. ©1964 by John Willey & Sons, Inc.

[142] Hebb, D.O (1949). The organisation of behaviour. Willy, New York.

[143] Hidaka K., H.Ohmori and A.Sano. Adaptive control design for linear time-varying system based on internal model principle. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p781-786.

[144] Hirsch, M.W. and S.Smale (1974). *Differential Equations, Dynamical Systems and Linear Algebra.* San Diego, CA: Academic Press.

[145] Hopfield J.J. (1982) Neural Networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci.USA*, Vol.79. 2554-2558, 1982.

[146] Ho D.W.C., Z.Ma. Multivariable internal model adaptive de-coupling controller with neural network for non-linear plants. Proceeding of the 1998 American Control conference (1998). Vol.1, p532-6.

[147] Horowitz R, B.Li and J.W.McCormick (1998). Wiener-filter-based Minimum Variance Self-tuning Regulation. *Automatica* Vol. 34 (5) p 531-545.

[148] Hornik, K., M.Stinchcombe and H.White (1989). Multilayer feed-forward networks are universal approximators. *Neural Networks.* Vol.2, p359-366.

[149] Hornik, K., M.Stinchcombe and H.White (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feed-forward networks. *Neural networks* Vol.3 (5), p551-560.

[150] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks* Vol.4, p251-257.

[151] Houpis Constantine H.Gary B.Lamount. (1992,1995). Digital Control Systems *Theory, Hardware, Software*. Mc Grraw Hill.

[152] Hunt K.J. and D. Sbarbaro (1991). Studies in neural networks for non-linear internal model control'. IEE proceedings Control Theory and Applications 138, p43-438.

[153] Hunt K.J. and D.Sbarbaro. Studies in artificial neural network based control. In Chapter 6 *Neural Network Applications in Control.* IEE Control Engineering Series 46 (1992) and Series 53 (1995). Edited by G.W.Irwin, K.Warwick and K.J.Hunt.

[154] Hunt K.J., D. Sharbaro, R.Zbikowski and P.J.Gawthrop (1992). Neural networks for Control Systems. *A SURVEY. Automatica* Vol. 28 (6), p1083-1112.

[155] Howard Demuth and Mark Beale (1992-93) Neural Network Toolbox MATLAB User's Guide. The Mathworks®.

[156] Hsu, J.A.Real (1997). Dual mode adaptive control with Gaussian networks. 36th IEEE conference of Decision and control. Vol. 4 p4032-4037.

[157] Hwang W.R. and W.E.Thompson (1993). An Intelligent Controller Design Based on Genetic Algorithms. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol.2 (4) p1266-1267.

[158] Iohvidov I.S. Hankel and *Toeplitz* Matrices and Forms. Algebraic Theory. (Edited by I.Gohberg and Traslated by G.Philp A.Thijsse) © Birkhauser Boston, 1982.

[159] Igor Aleksander and Hellen Morton (1995). An introduction to Neural Computing. International Thomson Computer press, ITP™.

[160] Isidori. Alberto Non-linear control systems. (2nd Edition). © Spring-Verlag. 1989. (1st Edition 1995).

[161] Irwin, G.W.,Warwick, K and K.J.Hunt. (1995). 'Neural Network Applications in Control'. IEE Control Engineering Series 53, London, UK

[162] Ilya Kraan and Peter M.M.Bongers. (1993). Control of a wind turbine using several linear robust controllers. *Proc.of the 32nd Conf.on Decision and Control San Antonio.* Vol. 2 of 4 p1928-1929.

[163] Jagannathan S., S.Commuri, F.L.Lewis. Feedback linearization using CMAC neural networks *Automatica* (UK) 1998. Vol. 134 (5) p547-557.

[164] Jeyasurya, B. (1994). ' Artificial neural networks for power system steady state instability evaluation'. *Electric Power Systems Research* Vol.29 (2), 85-90.

[165] Jin, L., P.N.Nikiforuk and M.M.Gupta (1994a). Absolute stability conditions for discrete-time recurrent neural networks. *IEEE Trans. on Neural Networks* Vol.5, p954-964.

[166] Jin L., and M.M.Gupta (1994a). Absolute stability conditions for discrete-time recurrent neural networks. *IEEE Trans. on Neural Networks* Vol.5, p954 – 964.

[167] Jin L., P.N.Nikiforuk and M.M.Gupta (1994b). Adaptive control of discrete-time nonlinear system using recurrent neural networks. *IEE Proc. on Control Theory and Applications.* Vol.141, p 169-176.

[168] Jin L., P.N.Nikiroruk and M.M.Gupta (1995). Approximation of discrete-time state space trajectories using dynamic recurrent neural networks. *IEEE Trans. on Automatic Control* AC 40, p1266-1270.

[169] Joao B.D. Cabrera and Kumpati S.Narendra (1999). Issues in the Application of Neural Networks for Tracking Based on Inverse Control. *IEEE Trans. on Automatic Control,* Vol. 44 (11), p2007-2027. **

[170] Jorgensen, C.C and Schley (1990) A neural network baseline problem for control of aircraft flare and touchdown. In W.T.Miller, R.S.Sutton and P.J.Werbos (Eds.) 'Neural Networks for Control'. London, MIT press Chapter 17.

[171] Jordon, M.I. R.A. Jacobs (1990). Learning to control and unstable system with forward modelling. In R.P.Lippmann. S.E.Moody and D.S.Touretzky (Eds.). 'Advances in Neural Information Proceeding Systems'. San Mateo: Morgan Kaufmann.

[172] Jones R.W. and M.T.Tham. (1987). Multivariable adaptive control: A *SURVEY* of methods and applications ch10 p280-307. Found in Johan O'Reilly. IEE Control Eng. series 32. Peter Peregrinus Ltd.

[173] Jun Song, Xiaoming Xu, Xing He (1998). A. stability based neural networks controllers design method. Control Cybern (Poland) Vol. 27, (1), p119-133.

[174] Johnsion R. L. Numerical Method: A Software Approach. © 1982 John Wiley & Sons, Inc John Wiley & Sons, Inc.

[175] Jorden, M.I. and R.A.Jacobs (1990). Learning to control an unstable system with forward modelling. In: R.P.Lippmann, S.E.Moody and D.S.Touretzhy (Eds.). 'Adavances in Neural information Processing Systems'. San Mateo: Morgan Kaufmann.

[176] Jose' L. Figueroa and Jose' A. Romagnoli. An algorithm for Robust Pole Assignment Via Polynomial Approach. IEEE Trans. on Auto. Control. (1994) Vol. 39 (4), p831-835.

[177] Juan M.Marti'n Sa'nchez , Jose' Rodellar. (1996). Adaptive Predictive Control *From the concepts to plant optimisation*. Prentice Hall International (UK) Limited. Hartnolls Limited, Bodmin, Cornwall 0-13-514861-8 629.836MAR.

[178] Kambhampati, C., J.D Mason, K. Warwick (2000). A stable one-step ahead predictive control of non-linear systems. *Automatica* Vol. 36, p 485-495.

[179] Kershenbaum L.S.and T.R.Fortescue. Implementation of on-line control in chemical process plants. *Automatica,* Vol. 17 *(6),* 777-788, 1981.

[180] Khalid, M and Omatu, S., (1992). " A neural network controller for a temperature control system". *IEEE Control System Magazines,* Vol. 12 (3) p 58-64.

[181] Kinnaert M. Generalised Prective control of multivariable linear systems. In *Proceedings of 26th Conference on Decision and Control.* p1247-1248, Los Angeles, CA, 1987.

[182] Kohonon, T. (1982). 'Self organised formation of topologically correct maps'. Biological Cybernetics 43, 59-69. *

[183] Kojima, Y.,Y.Izui, S.Kyomoto and T.Goda (1994). 'Voltage and reactive power-control using recurrent neural networks'. *Electrical Engineering in Japan* Vol.114 (4), 119-128.*

[184] Korenber M.J. and I. W. Hunter (1986). The Identification of Non-linear Biological Systems: LNL Cascade Models. Bilogical Cybernetics. Vol. 55, p125-134.

[185] Kosko, B. (1988). 'Bidirectional associative memories'. IEEE, Trans. on Systems Man and Cybernetics 18 (1), 49-60.*

[186] Krishna Kumar, K. (1994). Adaptive neuro-control for spacecraft altitude control. In: 'The Third IEEE Conferece on Control Applications'. Stratchclyde University, Glassgow. P1153-1158.

[187] Kulawski G.J. and M.A.Brdys.(1994). Adaptive Control of non-linear plants using neural networks: Application to a flux control in AC Drive system. IEE, International Conference Control '94' p1472-1477.

[188] Kwon Oh-Kyu, W.H.Kwon,K.S. Yoo,and M.J.Kim (1993). Receding Horizon LQG Controller Using Optimal FIR Filter with Control Input. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 2 of 4 p1292-1297.

[189] Lakmai C. Jain, Clarence W.de Silva (Editors). Intelligent Adaptive Control. *Industrial Applications.* ©1999 *by CRC Press LLC.*

[190] Landau I. D. (1979). Adaptive Control: The Model Reference Approach. *Series book:* Control and Systems Theory Vol. 8.

[191] Landau I. D. (1974). A *SURVEY* of Model Reference Adaptive Techniques—Theory and Applications. *Automatica*, Vol. 10, p353-379.

[192] Landau I. D. and H.M.Silveira (1979). A Stability Theorem with Applications to Adaptive Control. *IEEE Trans.on Automatic Control*, AC-24 (2) April 1979.

[193] Landau I. D. (1980). An extension of a Stability Theorem with Applications to Adaptive Control. *IEEE Trans.on Automatic Control*, AC-25 (4) August 1980.

[194] Ladau I.D (1993). Evolution of Adaptive Control. *Journal of Dynamic Systems, Measurement and Control.* Vol.115 p381-391.

[195] Lawson C.L and R.J.Hanson. *Solving Least Squares Problems.* Prentice Hall, Englewood Cliffs, NJ, 1974.

[196] Leigh J.R. (1992). Applied Digital Control, 2nd edition. Hemel Hemstead, UK, Prentice-Hall.

[197] Leigh J.R. (1983). Essentials of non-linear control theory. Peter Peregrinus Ltd.

[198] Leith D.J., and W.E. Leithead. Towards a theory of local model networks & Blended multiple model systems. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p509-605.

[199] Leshno *et al.*, 1993

[200] Lightbody Gordon and George W. Irwin (1997). Non-linear Control Structures Based on Embeded Neural System Models. *IEE Transaction on Neural Networks*, Vol. 8, (3), May 1997 p553-567.

[201] Lin, B.R. and R.G.Hoft (1994). 'Neural networks and fuzzy logic in power electronics'. *Control Engineering Practice* Vol.2 (1), 113-121.

[202] Liu G.P., (1996) V.Kadirkamanathan and S.A. Billings. Non-linear predictive via neural networks. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p746-751.

[203] Liu G.P. (2000), Neural–Learning Control of Non-linear Dynamical Systems (2000). IEE Control Workshop (Seminar), Learning Systems for Control. 26 May 2000 held at Austin Court Birmingham Ref: 00/069 p5/1-5/7

[204] Ljung Lennart. System Identification Toolbox MATLAB User's Guide ©1988-95 by The Mathworks®.

[205] ____________ (1977). Analysis of Recursive Stochastic Algorithm. IEEE Transaction on Automatic Control. Vol. AC 22 (4).

[206] ____________ (1987). System Identification Theory for the user. P T R Prentice-Hall.

[207] ____________ (1999). System Identification Theory for the user. P T R Prentice-Hall.

[208] L.Ljung and T.Soderstrom. *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, MA, 1983.

[209] Lahdhiri T. and A.T.Alouani.(1993). LQG/LTR Pitch Attitude Control of an Earth-Orbiting Spacecraft. *Proc.of the 32nd Conf.on Decision and Control San Antonio.* Vol. 1 of 4 p445-446.

[210] Lee, T.H.,W.K.Tan and M.H.Ang (1994). ' A neural network control system with parallel adaptive enhancements applicable to nonlinear servomechanisms'. *IEEE Trans.on Industrial Electronics.*Vol. 41 (3), 269-277.**

[211] Li Mingzhong, Wang Fuli (1997). Adaptive control of black box non-linear systems using neural networks. 36th IEEE conference of Decision and control Vol. 5, p4165-4170.

[212] Lianming Sun, Wenjiang Liu and Akira Sano. Least squares identification of Hammerstein Model based on over-sampling scheme. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p240-245.

[213] Loh, A.P., K.O. Looi and K.F.Fong (1995). ' Neural network modelling and control strategies for a PH process'. *Journal of Process Control.* Vol.5 (6), 355-362.

[214] MacFarlane A.G.J (1970). The return-difference and return-ratio matrices and their use in the analysis and design of multivariable feedback control systems. Proceedings of the Institution of Electrical Engineers, Vol. 117, 2037-49.

[215] Maciejowski J.M (1989). Multivariable Feedback Design. ©Addison-Wesley, UK-re-printed 1990, 1991,93 and 94.

[216] Madsen P.P (1995). Neural Network for optimization of existing control systems. *In Proc. 1995 IEEE International Conference on Neural Networks,* Perth, Australia, p.1496-1501.

[217] Mahdi Riyad Issa (1996). Optimal PI-Lead Controller Design. IEEE Computer Society. *Proceeding of the twenty-eighth, South-eastern Symposium on System Theory.* p364-368.

[218] Mahseredjian Jean *et al.* (1998). A link between EMTP and MATLAB for user-defined modelling. ©IEEE, Transactions on Power Delivery, Vol. 13 (2), p667-673.

[219] Marino Riccardo & Patrizio Tomei. Non-linear Control Design *Geometric, Adaptive, Robust.* © Prentice Hall International (UK) Limited 1995.

[220] Marino Riccardo & Patrizio Tomei (1999). An Adaptive Output Feedback Control for a Class of Nonlinear Systems with Time-Varying Parameters. *IEEE Trans. on Automatic Control,* Vol. 44 (11),

p2190-2194.

[221] Marios M.Polycarpou (1998). Indirect Adaptive Non-linear control of Drug Delivery Systems. *IEE Trans.on Automatic Control* Vol. 43 (6) June 1998 p 849-854.

[222] Marios M.Polycarpou (1998). Indirect Adaptive Non-linear control of Drug Delivery Systems. *IEE Trans.on Automatic Control* Vol. 41 (3) March 1998 p447-450.

[223] Marriott S. and R.F.Harrison. A Self-organising adaptive neuro-controller using reinforcement learning. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p1113-1118.

[224] Mats F.Sa°gfors et al (1997). State-space solution to the periodic multirate H_∞ Control problem: A lifting approach. *36th IEEE Conferences on Decision and Control*. © 1997 by the Institute of Electrical and Electronic Engineers, Inc. Vol. 3, p-2061-2066.

[225] Mayne Q. David and Hannah Michalska. Receding Horizon Control of Non-linear Systems. IEEE Trans. on Automatic Control, Vol. 35 (7), July 1990 p814-824.

[226] McClelland T.L and D.E.Rummelhart and the PDP Research Group. Parallel Distributed Processing. MIT press, Cambridge, MA, 1986.

[227] McCulloch, W. and W.Pitts (1943). 'A logical calculus of the ideas immanent in nervous activity'. Bulletin of Mathematical Biophysics 5, p115-133. *

[228] Michael Schiebe and Saskia Pferer (Editors). Real-Time Systems Engineering and Application, 1992, by Kluwer Academic Publishers.

[229] Michael L.Honig, David G.Messerschmitt (1984). Adaptive Filters. Adaptive Filters, Structures Algorithms and Applications. Kluwer Academic Publishers.

[230] Michael D. Lemmon (1999), (Guest Editorial), Special Section on Neural Networks in Control, Identification and decision making. IEEE Trans. on Automatic Control Vol. 44 (11) p1993-1994.

[231] Michie, D. and R.A. Chambers (1968). Boxes: An experiment in adaptive control. In J.T.Tou and R.H.Wilcox (Eds.). 'Machine Intelligence'. Edinburgh: Oliver and Boyd. Chapter 2, p137-152.*

[232] Mietek A. Brdys and Grzegorz J. Kulawski (1999). Dynamic Neural Controllers for Induction Motor. IEEE Transactions on Neural networks, Vol. 10, (2), p340-355.

[233] Miller Thomas W., III, Richard S.Sutton, and Paul J. Werbos. (1990). *Neural Networks for Control*. The MIT Press Cambridge, Massachusetts London, England.

[234] Moody, J. (1989). Fast-learning in multi-resolution hierarchies. In D.S.Touretzky (Eds). 'Advances in Neural Information Processing Systems'. Morgan Kaufmann. p 29-39. Vol. 1. *

[235] Monostori, L. And D.Barschdorff (1992). 'Artificial neural networks in intelligent manufacturing.' *Robotics and Computer-Integrated Manufacturing* 9 (6), p421-437.

[236] Morari, M.and E.Zafiriu (1989). *Robust Process Control.* Prentice Hall International.

[237] Mendes, M. An on Line Adaptive Control Method. *Automatica*, Vol. 7, p 323-332 (1971).

[238] Narendra and Annaswamy (1989). Stable Adaptive Systems. Prentice Hall.

[239] Narendra, K.S. and Parthasarathy, K., " Identification and Control of dynamic systems using neural networks, " *IEEE Trans. on Neural Networks*. Vol. 1 p 4-27, 1990.

[240] Narendra, K.S. and Parthasarathy, K. (1991) " Gradient methods for the optimisation of dynamical

systems containing neural networks, " *IEEE Trans. on Neural Networks.* Vol. 2 (2), p 252-262.

[241] Narendra Kumpati S. and Snehasis Mukhopadhyay (1997). Adaptive Control Using Neural Networks and Approximate Models. IEEE Transactions on Neural Networks, Vol. 8 (3) p475-485.

[242] Nash Peter. (1981). Systems Modelling and Optimisation. IEE Control Engineering Series 16. Peter Peregrinus Ltd.

[243] Naser N., S.Zein-Sabatto (1998). An intelligent controller design based on system parameter estimation. *Proc.of Thirtieth (30th) South-eastern Symposium on system theory.* p172-175.

[244] Neto A. Trofino, B.Brogliato, I.D. Landau. (1993). LQG Controllers subject to passive constraint. *Proc.of the 32nd Conf.on Decision and Control San Antonio.* Vol. 2 of 4 p11531-1535.

[245] Newell R.B.and P.L.Lee. (1989). Applied Process Control. *A case study.* By Prentice Hall of Australia Ltd.

[246] Norbert Wiener. Non-linear Problems in Random Theory. © 1958 by Massachusetts Institute of Technolgy.

[247] Norton J.P. An Introduction to Identification. © 1996 by Academic Press Inc. (London) Ltd.

[248] Ng G.W. and P.A.Cook (1996). Neural network in control of systems with unknown and varying time delays. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, p188-193.

[249] Ng G.W. and P.A.Cook (1996a). 'Integrated gradient and least squares algorithm for fast learning and control applications'. UMIST, Control Systems Centre report number 846.*

[250] Ng G.W. and P.A.Cook (1996b). 'Local convergence and stability analysis of neural network controller. UMIST, Control Systems Centre report number 845.*

[251] NG G.W. (1997). Application of Neural Networks to Adaptive Control of Non-linear Systems. UMIST Control Systems Centre Series, by Research Studies Press Ltd. Research Studies Press LTD, England.

[252] Nguyen, D. And B.Widrow (1989). The truck backer-upper: an example of self-learning in neural networks'. *Proceedings of international joint conference on Neural networks.* 2, p11357-11363.

[253] Nguyen, D. And B.Widrow (1990). 'Neural networks for self-learning control systems'. *IEEE Control Systems Magazine* 10(3), p18-23.

[254] Nusret Tan and Derek P.Atherton (2000). Stability and performance analysis in an uncertain World. IEE, Computing and Control Engineering Journal. April 2000. p91-101

[255] Ogata Katsuhiko (1970). Modern Control Engineering. © 1970 by Prentice-Hall, Inc., Englewood Cliffs, N.J., U.S.A.

[256] ____________ (1994). Solving Control Engineering Problems with MATLAB. © Prentice-Hall, Inc.

[257] ____________ (1994). Designing Linear Control Systems with MATLAB. © Prentice-Hall, Inc.

[258] Omid Omidvar and David L.Elliott (1997). Neural Systems for Control. Academic Press.

[259] O'Reilly Johan (1987). Multivariable control for industrial applications. IEE Control Engineering Series 32. Peter Peregrinus Ltd.

[260] Ohno M., M.Takahama, T.Kimura and E.Tokuda. (1993). H_∞ Control Design Method Combined

with Exact Model Matching -Design of Longitudinal Robust Flight Control System. *Proc.of the 32nd Conf.on Decision and Control San Antonio*, Vol. 1 of 4, p447-448.

[261] Oubrahim R., and F.Leonard. PID tuning by a composed structure. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p1333-1338.

[262] Alexander G. Parlos (1994). Application of the Recurrent Multilayer Perceptron in Modelling Complex Process Dynamics. IEEE Transactions on neural networks, Vol.5 (2), March 1994 p257-266.

[263] Patino H.D and Derong Liu (2000). Neural Network-Based Model Reference Adaptive Control System. IEEE Trans. on systems, Man. And Cybernetics-part B, Vol. 30 (1), p198-204.

[264] Park, J. and I.W.Sandberg (1991). Universal approximation using radial basis function networks. *Neural computation* Vol.3, p246-257.

[265] Paganini Fernando (1993). Set Descriptions of White Noise and Worst Case Induced Norms. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 4 of 4 p3658-3663.Paganini Fernando. (1996). A Set Based Approach for White Noise Modelling. © *IEEE Trans. on Auto.Cont.,* Vol. AC-41, (10) p1453-1465.

[266] Petersen Ian R. (1993). Guaranteed Cost LQG Control of Uncertain Linear Systems. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 3 of 4 p2020-2025.

[267] Parisini T. and R.Zoppoli (1996), *IEEE Trans. on Automatic Control,* Vol. 41(6) June 1996 p 889-894.

[268] Pham D.T. and X.Liu. (1995). 'Neural Networks for Identification, Prediction and Control. © Springer - Verlag Berlin Herdelberg New York.

[269] Philippe De Wilde. (1997). Neural Network Models. 2nd Edition © Springer-Verlab London Limited 1997 Athenaeum Press Ltd., Gateshead, Tyne and Wear 69/3830-543210 (3-540-76129-2) (IEE Library 519.7:616.8 DE).

[270] Pickhardt R and H. Unbehauen. Adaptive control of plants subject to changes of structure and parameters using a multi-model approach. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p656-661.

[271] Porter, W.A.and W.Liu (1994). Neural feedback controller. In 'IEEE Southeascon Conference Proceedings'. IEEE, Piscataway, N.J.USA. p485-489.*

[272] Potvin Andrew F. (1993). Non-linear Control Design Toolbox MATLAB User's Guide Mathworks®.

[273] Philip G.Gallman (1975). An Iterative Method for Identification of Non-linear Systems Using a Uryson Method. *IEEE.Trans.on Automatic Control* (1975). p771-775.

[274] Primoz Potocnick and Igor Grabec (2000). Adaptive self-tuning neuro-control. Mathematics and Computers in Simulation 51, Elsevier, p201-207.

[275] Psaltis, D., Sideris, A., and Yamamura, A.A (1988). " A multi-layered neural network controller." *IEEE Control System Magazines,* p17-20.

[276] Puskorious G.V. and Lee A. Feldkamp (1994). Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks. IEEE Transactions on neural networks, Vol.5 (2), March 1994 p281-297.

[277] Puskorious G.V. and L.A. Feldkamp and L.I.Davis Jr., (1996). Dynamic neural network methods applied to on vehicle idle speed control. *Proceedings of the IEEE.* Vol.84 p1407-1420.

[278] Raymond B.Sepe. (1993). Robust LQG Position Control of a Flexible Structure Using Participation Factors. *Proc.of the 32nd Conf.on Decision and Control San Antonio.* Vol. 2 of 4 p11396-1401.

[279] Rajbman N.S (1976). The application of Identification Methods in the U.S.S.R. A-*SURVEY. Automatica* Vol. 12, p73-95.

[280] Rivals I., L.Personnaz (1998). A recursive algorithm based on the extended Kalman filter for the training of feedforward neural models. Neuro-computing (Netherlands). Vol. 20 (1-3), p279-294.

[281] Ruiz A., D.H.Owens and S.Townley. Learning periodic signals with recurrent neural networks. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p1131-1136.

[282] Robert L.Borrelli, Courtney Coleman and William E.Boyce. Differential Equations Laboratory Workbook: *A collection of Experiments, Explorations and Modelling Projects for the Computer.*

[283] Robert L.Hey. Neural Network Principles. ©1994 by Prentice-Hall, Inc. (*Note: Good little book, this book concentrate mostly on biological view point)*

[284] Ren Wei.(1993). The Self-Tuning of Stochastic Adaptive Pole Placement Controllers. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 2 of 4 p1581-1586.

[285] Raymond B.Sepe. Robust LQG Position Control of a Flexible Structure Using Participation Factors. *Proc.of the 32nd Conf.on Decision and Control San Antonio*, Dec.1993, Vol. 2 of 4 p11396-1401.

[286] Robert Mahony, Uwe Helmke and Johm Moore (1993) .Pole Placement Algorithms for Symmetric Realisations. *Proc.of the 32nd Conf.on Decision and Control San Antonio*, Dec.1993, Vol. 2 of 4 p1355-1358.

[287] Robinson C. and N.Mort (1996). A neural network solution to the problem of Frost Prediction. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, p136-139.

[288] Robert Molten Gray (1972). On the Asymtotic Eigenvalue Distribution of *Toeplitz* Matrices. IEEE Trans. on information Theory, Vol. IT-18 (6), November 1972.

[289] Rosenblatt F. The perceptron: a probabilistic model for information storage and organisation in the brain. Psychol. Rev., 1958 Vol.65 pp 386-408.

[290] R*ü*ger S.M., A Class of Asymptotically Stable Algorithms for Learning -Rate Adaptation. Algorithmica (1998) 22, p198-210.

[291] Rumelhart. D.E. and McClelland, J.L. (Eds.) (1986). Parallel Distributed Processing: explorations in the microstructure of cognition. Vol. 1. London: MIT press. *

[292] Safonov M.G. and Athans M. (1977). Gain and phase margins of multiloop LQG regulators. *IEE Trans.on Automatic Control*, AC-22, p173-179.

[293] Safonov M.G., Laub A.J.and Hartmann G.L. (1981). Feedback properties of multi-variable systems: The role and use of the return difference matrix. *IEE Trans.on Automatic Control*, AC-26, 47-65.

[294] Sama, T (1992). " Neuro-control: concepts and applications," 1992 *IEEE Intl. Conf. On Systems, Man and Cybernetics*, Vol. 1, p369-74, Chicago , IL, 1992.

[295] Sanner, R.M. and J.-J.E.Slotine (1991). Stable adaptive control and recursive identification using radial gaussian networks. In: *Proc. of the 30th Conference on Decision and Control, Brighton, England. Pp.2116-2123.*

[296] Sandoz David J., Matthe J.Desforges, Barry Lennox and Peter R.Goulding (2000). Algorithms for industrial model predictive control. Computing & Control Engineering Journal Vol. 11 (3) p125-134.

[297] Sbarto, D., D.Neumerkel and K.Hunt (1993). 'Neural control of a steel rolling mill'. *IEEE Control System Magazines.* 13(3), p70-75.

[298] R.Scattolini and N.Schiavoni. Generalized minimum variance control of MIMO systems- stability result. *Automatica,* Vol.23 (6), 797-799, 1987.

[299] Ssu-HsinYu, A.M.Annaswamy. Stable neural controllers for non-linear dynamic systems. *Automatica* (UK) (1998). Vol. 134 (5), p641-50.

[300] Schwarz H (1996). Changing the unstable zero dynamics of non-linear systems via parallel compensation. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, p1226-1231.

[301] Sastry Shankar and Marc Bodson (1989). Adaptive Control: Stability, Convergence and Robustness. Prentice-Hall International Editions.

[302] Sastry P.S., G.Santharam and K.P.Unnikrishnan (1994). Memory Neuron Networks for Identification and Control of Dynamical Systems. IEEE Transactions on neural networks, Vol.5 (2), March 1994 p306-319.S.L.Shah, C.Mohtadi and D.W.Clarke. Multivariable adaptive control without prior knowledge of the ideally matrix. *Syst.Control Lett.,* Vol.9, 295-306, 1987.

[303] Sira H.J. Ramirez and S.H.Zak (1991). The adaptation of perceptrons with application to inverse dynamic identification off unknown dynamic systems. *IEEE Trans.Syst.Man.Cybern*. 1991 Vol.21, pp634-643.

[304] Sjoberg, J., H.Hjalmarsson and L.Jung. (1994). Neural networks in system identification. In: *Proc. 10th IFAC Symposium on System Identification SYSID '94' Copenhagen.* Vol. 2, p49-72.

[305] Sjoberg, J. and Ljung, L. (1995). Over training, regularization and searching for minimum in neural networks. Int. Journal of Control, Vol. 62 (6), p1391-1408.

[306] Sjoberg, J., Zhang, Q., Ljung, L.,Benveniste, A.,B.Delyon, P.-Y.Glorennec, H.H. and Juditsky, A. (1995). Non-linear black-box modelling in system identification: a unified overview. *Automatica,* Vol. 31(12), 1691-1724.

[307] Sklansky Jack. Learning Systems for Automatic Control. IEEE Trans. on Automatic Control (1966). Vol. AC-11 (1) p6-19.

[308] Slotine, J.J.E and W.Li (1991). *Applied Nonlinear Control.* Prentice Hall.

[309] Slotine, J.J.E. and R.M. Sanner (1993). Neural networks for adaptive control and recursive identification: A theoretical framework. In: *Essays on control: perspectives in the theory and its applications* (H.Trentelman and J.Willems. (Eds). Vol.14 of *Progress in Systems and Control Theory.* p381-436. Birkhauser!

[310] Smith C.A., K.J.Burnham, D.J.G.James. Enhanced self-tuning control with application to a combined heat and power system. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996

p1064-1069.

[311] Smolensky, P. (1986). Harmony theory. In: D.E.Rumelhard and J.L.McClelland (Eds). 'Parallel Distributed Processing: explorations in the microstructure of cognition'. London: MIT press. Cambridge MA. Chapter 6, Vol. 1 p194-281.

[312] Song, Q., J. Wilkie and M.J.Grimble (1994). An integrated robust/ neural controller with gas turbine applications. In: 'The 3rd IEEE Conference on Control Application'. Stratchlyde University Glassgow. p411- 414. *

[313] Song, Q. and M.J.Grimble (1997). Design of a Multivariable Neural Controller and its Application to Gas Turbines. Journal of Dynamic Systems, Measurement and Control. Vol. 119 p565-567.

[314] Sorensen Ole (1993). Neural Networks performance system identification for control applications. In 3rd Int.Conf.on Artificial Neural networks, Brighton, UK.

[315] N.R.Sripada and D.G.Fisher. Improved least squares identification. *Int. J.Control.***46,** pp1889-1913, 1987.

[316] N.R.Sripada and D.G.Fisher. Improved least squares identification. *Int. J. Control.* Vol.46, 1889-1913, 1987.

[317] Stein G.and Atans M.(1987). The LQG/LTR procedure for multivariable feedback control design. *IEE Trans.on Automatic Control*, AC-32, 105-14.

[318] Sontag, E.D. (1993). Neural networks for control. In: *Essays on control: perspectives in the theory and its applications* (H.Trentelman and J.Willems. (Eds). Vol.14 of *Progress in Systems and Control Theory.* Pp.339-380. Birkhauser.!

[319] Sontag, E. 'Recurrent neural networks: some systems theoretic aspects', in Karny, M., Warwick K. And Kurkova V (Eds.) 'Dealing with complexity: a neural networks approach' (Springer, 1998).

[320] Ssu-HsinYu, A.M.Annaswamy (1998). Real neural controllers for non-linear dynamic systems Eng. *Appl. Artif. Intel.* (UK) 1998. Vol.11, (3), p401-409.

[321] Suykens Johan A.K., Joos P.L.Vandewalle, Bart L.R.De Moor. Artificial Neural Networks for Modelling and Control of Non-Linear Systems. © Kluwer Academic Publishers Kluwer Academic Publishers Boston / Dordrecht/ London, 1996.

[322] Suykens Johan A.K., Bart L.R.De Moor and Joos Vandewalle (1995). Nonlinear system identification using neural neural state space models, applicable to robust control design. *Int.J.Control,* 1995, Vol. 62 (1), p129-152.

[323] Taylor C.J., P.C.Young, A.Chotai, W.Tych and M.J.Lees. The importance of structure in PIP control design. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p1196-1201.

[324] Thomas J. Moir and Michael J.Grible (1984). Optimal Self-tuning Filtering, Prediction and Smoothing for Discrete Multivariable Processes. IEEE Trans. on Auto. Control, AC-29 (2).

[325] Thapa B.K (1998) Internal Report: Qualifying Progress Report for 1997/98. Aston University. School of Engineering and Applied Science.

[326] Thapa, B.K and T.Earthe_Gould (1999). Back-propagation Neural Network Enhanced Non-linear System Identification and Control. *IEE Sri-Lanka Centre 5th Annual Conference.* (Refreed and resubmitter

for the proceeding of the IEE Sri-Lanka on 20 Aug.2000. To be appear

[327] Thapa B.K., B.Jones and Q.M.Zhu (2000). Non-linear control with neural network. KES'2000. 4th International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies. 30Aug-1 Sep.2000, Brighton, UK p868-873.

[328] Thomas M. and A.Wyner. (1987). Lecture Notes in Control and Information Sciences. © Springer-verlag Berlin, Heidelberg. Springer-verlag Berlin, Heidelberg in Germany 3-387-18055-9(US).

[329] Tolat, V.V., and Widrow, B., " An adaptive 'broom balancer' with visual inputs', *Proc. of the Int. Conf. On Neural Networks,* II, 641-647, IEEE Press, New York, July 1988.

[330] Tolle, H. (1994). Learning control and its place in automation. In: 'Intelligent Systems Engineering'. Institute of Electrical Engineers, London, p9-36. IEE Conference pub. No. 395.

[331] Tsypkin Ya. Z. Self-Learning What Is it? *IEEE Trans.on Automatic Control*, Dec. 1968. Vol. AC-13, (6) p 608-612.

[332] Tsypkin Ya.Z. Adaptation and Learning in Automatic Systems. Mathematics in Science and Engineering. Academic Press. New York and London. 1971 Vol. 73. (English Translation by Z.J.Nikolic).

[333] Tuffs P.S. Software aspects of self-tuning control. Ch8. p157-180. **In**: K.Warwick: Implementation of self-tuning controllers. © 1988 Peter Peregrinus Ltd.

[334] Turtle, D.P. and P.H.Phillipson (1971). Simultaneous Identification and Control. *Automatica*, Vol. 7, p445-453.

[335] Ramacher Ulrich, VLSI design of neural networks. ©1991 by Kluwer Academic Publishers.

[336] VanDoren V.J., Advanced control software goes beyond PID Periodical title: Control Eng, (USA)(1998).Vol. 45 (1) p73-4, 76,78,

[337] Verest S.M., D.S.Wall, A.V.Kuntsevich and S.Hermsmeyer. Using GBT for MATLAB Version 5.1, In Identification and control. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p216-221.

[338] Vicken Kasparian and Celal Batur (1998). Model reference based neural network adaptive controller. ISA Transactions 37, p21-39.

[339] Wang, L. (1994). *Adaptive fuzzy systems and control: design and stability analysis.* PTR Prentice Hall.

[340] Wang S (1998). An insight into the standard backpropagation neural network model for regression analysis. *Omega* (UK), Vol. 26 (1), p133-40.

[341] Wang, D.H. and C.B.Soh (2000), Adaptive neural model-based decentralized predictive control. *Int. Journal of Systems Science*, Vol. 31 (1), p119-130.

[342] Wang Jin & Gao Wenzhong (1997). Adaptive poleplacement Control Algorithm for nonlinear systems. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems Vol.5 (2) 139-146. (c) World Scientific Publishing Company.

[343] Wams, B. M.Ayala Botto, T.J.J.Van den Boom and J.Sa' da Costa (1998). Training Neural networks for robust control of non-linear MIMO systems. *UKACC Int. Conf. on CONTROL*'98. Conf. Publication No: 455 ©IEE, 1998 p241-246.

[344] Waltz, M.D. and K.S.Fu (1965). 'A heuristic approach to reinforcement learning control systems'. *IEEE.Trans.on Automatic Control* AC-10 (4), p390-398.*

[345] Warwick, K., C.Kambhampati, P.Parks and J.Mason (1995). Dynamic systems in neural networks. In: K.J.Hunt, G.Irwin and K.Warwich (Eds.). 'Neural networks Engineering in Dynamic Control System'. Springer-Verlag London Limited. Ch.2 p27-41.

[346] Warwick, K (1994). Neural networks for control: counter arguments. In*: Proc. Int.Conf.CONTROL'94*, Warwick, England. p94-99.

[347] Warwick, K. (1995). A critique of neural networks for discrete-time linear control. *Int.J.Control,* 1995, Vol.61 (6), p1253-1264.

[348] Warwick, K., Ekwue, A., and Aggarwal, R. (Editors) (1997): 'Artificial Intelligence Techniques in Power Systems. (IEE Power Engineering Series 22).

[349] Warman D. and R.Roddis. Design of an Electro-Hydraulic Multivariable Control System Teaching Facility. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p36-41.

[350] Watanabe, K., K.Hara, S.Koga and S.G.Tzafestas (1995). 'Fuzzy neural network controllers using mean-value based functional reasoning'. *Neurocompuing*. Vol. 9 (1), 39-61.

[351] Wellstead P.E.and M.B.Zarrop. Self-tuning Systems: Control and Signal Processing. © 1991 by John Wiley &Sons Ltd.

[352] Werbos, P.J. (1990b). Overview of designs and capabilities. In: W.Thomas, Miller, R.S.Sutton and P.J.Werbos (Eds) 1 'Neural Networks for Control ©1990 Massachusetts Institute of Technology A Bradford Book. The MIT Press. Chapter 2, p59-65.

[353] Werner H. Genelized Sampled-Data Hold Functions for Robust Multivariable Tracking and Disturbance Rejection. 36th *IEEE Conferences on Decision and Control.* © 1997 by the Institute of Electrical and Electronic Engineers, Inc. Vol. 3, p2055-2060.

[354] Willems Jan C. (1993). LQ-Control: A Behavioural Approach. *Proc.of the 32nd Conf.on Decision and Control San Antonio*. Vol. 4 of 4 p3664-3668.

[355] Willems Jan C. and Sanjoy K.Mitter. (1971). Controllability, Observability, Pole Allocation, © *IEEE Trans. on Automatic Control,* Vol., AC-16, (6), p582-595.

[356] Willems, T.M. (1993) *Neural networks in Control?* Eindhoven: Technishe Universiteit Eindhoven.

[357] Widrow, B. and F.W.Smith (1963), 'Adaptive neural networks and their applications'. International Journal of Intelligent Systems 8(4), 453-507. *

[358] Widrow B. and M.A.Lehr (1990). Thirty years of adaptive neural networks: Perceptron, madaline and back-propagation. *Proc. IEEE* 1990, Vol.78, pp 1415-1442.

[359] Widrow B. and Marcian E.Hoff (1960). Adaptive switching circuits. 1960 IRE WESCON Convention Record, New Yourk: IRE, p 96-104. (Can be found in Neural Computing: Foundation of Research edited by James A. Anderson and Edward Rosenfeld 4th printing (1988,89).

[360] Wigren T. and Anders E. Nordsjo (1999). Compensation of the RLS Algorithm for output Nonlinearities. *IEEE Trans. on Automatic Control,* Vol. 44 (10), 1999 p1913-1918.

[361] Williams, R.J., "Adaptive state representation and estimation using recurrent connectionist networks.

In T.Miller *et al.* (Eds). *Neural networks for Control*. MIT press, Cambridge, MA, 1990.

[362] Williams, R.J., and J.Peng, (1990). "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation,* Vol. 2, p.490-501.

[363] Williams, R.J., and D.Zipser (1989). "A learning algorithm for continually running fully recurrent network ," *Neural Computation,* Vol. 1, p.270-280.

[364] Xiao D.Ji and Babajide O.Familoni. A Diaogonal Recurrent Neural Network-Based Hybrid Direct Adaptive SPSA Control System. *IEEE Trans. on Automatic Control,* Vol. 44 (7), 1999 p1469-1473.

[365] Yang Y.Y and D.A.Linkens (1999). Modelling of the flow stress using BP network. IEEE, Proceedings of the second International conference on intelligent processing and manufacturing of materials (IPMM), Vol. 2. p755-761.

[366] Yang, Y.Y. and D.A.Linkens (1994). 'Adaptive neural-network based approach for the control of continuously stirred tank reactor'. *IEE Proceedings Control Theory and Applications.* 141 (5) p 341-349

[367] Yun-Chung. Jie Huang (1998). Solving the non-linear regulator equations by a single layer feed-forward neural network. *Comput. Ind. Eng.* UK, Vol. 35 (1-2), p359-362.

[368] YukseY.Onder *et al.* (1971). Observers for linear Multivariable Systems with Application. *IEEE Trans. on Auto.Cont.,* Vol., AC-16, (6), p603-613.

[369] Yuri P.Grishin. A Robust Limited Memory Recursive Filter. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p1087-1088.

[370] Zalama, E.,P. Gaudiano and J.L.Coronado (1995). 'A real-time, unsupervised neural network for the low level control of a mobile robot in a nonstationary environment'. *Neural Networks* Vol.8 (1), p103-123.

[371] Zhuang M. and D.P.Atherton. CAD of Non-linear Controllers for Non-linear Systems. *UKACC Int. Conf. on CONTROL*'96. Conf. Publication No: 427 ©IEE, 1996 p545-550.

[372] Zhuang M. and D.P.Atherton (1994). PID Controller designs for TITO system. © *IEE Proc. of Control Theory and Application*, Vol. 141, No.2 March. p111-120.

[373] Zhu Q. M. (1989). Identification & Control of Non-linear Systems. PhD Thesis. Warwick University, UK.

[374] Zhu Q. M. and K. Warwick (1991). Deadbeat Controller for a class of non-linear systems. Mita Press Tokyo Japan. p101-116.

[375] Zhu Q.M. and S.A. Billings (1996). Fast orthogonal Identification of non-linear stochastic models and radial basis function neural networks. *Int. J. Control*, 1996, Vol. 64 (5), 871-886.

[376] Zhu Q. M. and S.A. Billings (1991). Recursive Parameter Estimation for Non-linear Rational Models. *Journal of Systems Engineering*, p63-76.

[377] Zhu, Q.M., Z. Ma. and K. Warwick, (1999). Neural Network enhanced generalised minimum variance self-tuning controller for non-linear discrete time systems. IEE Proc. Control Theory Application. Vol. 146 (4) July 1999 p319-326.

[378] Zhu, Q.M. (2000). Intelligent PID controller for non-linear dynamic systems. IEE Control Workshop (Seminar), Learning Systems for Control. 26 May 2000 held at Austin Court Birmingham, UK, Ref: 00/069. p11/1-11/3.

[379] Special Issue on neural networks. *IEEE Control Systems Magazine* April 1988 Vol. 8 (2), 1989 Vol. 9 (3), 1990, Vol. 9 (3), 1992 Vol. 12 (2).

[380] Special Feature (2000): *Intelligent Sensors.* Computing and Control Engineering Journal. Vol.11 (5). Oct. 2000.

[381] ftp://ftp.sas.com/pub/neural/FAQ2.html

[382] www.mathworks.com

[383] http://forum.iee.org.uk/forum/library/view.cgi/2001 05/grimble.html

APPENDIX A: NEURAL NETWORK GLOSSARY OF TERMS

Activation function: (See Transfer Function)

Adaptive: Adaptive neural networks classify patterns. Input data similar to previously seen patterns are classified as one of them. Patterns not similar to previous ones have; new class of patterns created for them.

Adaptive Resonance Theory (ART): A type of neural network model trained without supervision, which is used in pattern classification problems. See [DARPA].

ADALINE: an acronym for a linear neuron: ADAptive LINear Element. (Adaptive Linear Neurone): See also "Perceptron." "Multi-layer Perceptron" and "Single-layer Perceptron."

Adaption: a function that proceeds through the specified sequence of inputs, calculating the output, error and network adjustment for each input vector in the sequence as the inputs are presented.

Adaptive learning rate: a learning rate that is adjusted according to an algorithm during training to minimize training time.

Adaptive filter: a network that contains delays and whose weights are adjusted after each new input vector is presented. The network "adapts" to changes in the input signal properties if such occur. This kind of filter is used in long distance telephone lines to cancel echoes.

Algorithm: A list of rules or equations that determine how a neural network operates and trains.

Architecture: a description of the number of the layers in a neural network, each layer's transfer function, the number of neurons per layer, and the connections between layers. (Also see Topology)

Artificial neural networks (ANNs): ANNs are simulations of how it is thought the animal brain operates. It has been found that these simulations possess powerful pattern recognition and prediction abilities - human-like qualities.

Association: An associative neural network will recall the closest "stored" training pattern when presented with a similar, but possibly noisy, input pattern.

Back-propagation (BP) learning: Another way to define BP learning - a learning rule in which weights and biases are adjusted by error derivative (delta) vectors backpropagated through the network. BP is commonly applied to feedforward multilayer networks. Sometimes this rule is called the generalized delta rule.

Back-tracking search: linear search routine which begins with a step multiplier of 1 and then backtracks until an acceptable reduction in the performance is obtained.

Batch: a matrix of input (or target) vectors applied to the network "simultaneously". Changes to the network weights and biases are made just once for the entire set of vectors in the input matrix.

Batching: the process of presenting a matrix (batch) of input vectors for simultaneous calculation of a matrix of output vectors and/or new weights and biases.

Bayesian framework: assumes that the weights and biases of the network are random variables with specified distributions.

BFGS quasi-Newton algorithm: a variation of Newton's optimization algorithm, in which an approximation of the Hessian matrix is obtained from gradients computed at each iteration of the algorithm.

Bias: a neuron parameter that is summed with the neuron's weighted inputs and passed through the neuron's transfer function to generate the neuron's output.

Bias vector: a column vector of bias values for a layer of neurons.

Brent's search: a linear search, which is a hybrid combination of the golden section search and a quadratic interpolation.

Bi-directional Associative Memory: A type of neural network models, which is an associative version of the Hopfield Network.

Binary output: An output that can only take one of two values. For example in a control system an output neuron that indicated a fault had occurred would have a binary output (cf. continuous output).

Boltzmann Machine: A type of supervised neural network learning algorithm in which network states are determined by "simulated annealing." Boltzmann machines use noise process to find the global minimum of a cost function.

Boundary Contour System (BCS): A type of neural network algorithm used in image segmentation problems.

Cellular Automata: A mathematical formalisation for parallel processes. Specifically a cellular automaton is a graph whose nodes are finite-state machines (thus the under lying graph or "space" of a given cellular automaton is considered to be fixed; it cannot be altered by any of its nodes). The operation of a cellular automaton is determined by information passed between those nodes that are connected (in most cases the interconnections between nodes pass information bi-directionally).

Classification: an association of input vector with a particular target vector.

Competitive layer: a layer of neurons in which only the neuron with maximum net input has an output of 1 and all other neurons output 0. Neurons compete with each other for the right to respond to a given input vector.

Cochlea Chip: An analogue VLSI circuit modelled after the biological cochlea (a part of mammalian ear).

CMAC (Cerebellar Model Articulated Controller): A type of neural network model adaptively forms complex non-linear maps and is typically used in motor control problems and defined by a redundant but direct, one-to-one, feed-forward connection topology. [Barto *et al.*]

Competitive learning: An unsupervised learning algorithm in which groups of processing elements in a neural network compete among themselves to respond to a set of stimulus input patterns. The winner within each group is the one whose connections make it respond most strongly to the pattern; the winner then adjusts its connections slightly toward the pattern that it won [Darpa].

Competitive transfer function: accepts a net input vector for a layer and returns neuron outputs of 0 for all neurons except for the "winner," the neuron associated with the most positive element of the net input *n* [Demuth 1996].

Competitive network: A competitive neural network has neurons, which compete with their neighbours such that only one neurone will respond to a particular input pattern. See also Kohonen.

Computational Maps: Two-dimensional arrays (often stacked along a third dimension) have locally interconnected processing elements that represent variables or objects by the position and pattern of activity on their surfaces. Computational maps exhibit properties of topological self- organisation, self-optimisation and fault tolerance.

Confusion matrix: A visual indication of how well the neural network has trained. Two axes represent the desired and actual responses and a dot is placed for each fact in the training or test set. For a neural network that is correctly trained the confusion matrix will be a straight line at 45^o.

Connection: a one-way link between neurons in a network. (i.e., A path from one neurone to another to transfer information. Also called synapses, which are often associated with weights that determine the strength of the signal that is transferred).

Connection strength: the strength of a link between two neurons in a network. The strength, often called weight, determines the effect that one neuron has on another.

Connection per second (CPS): This is the measure of speed of operating for a neural network. Neural network simulation software and hardware often state this number for learning mode and run mode. Manufacturers often exaggerate these numbers and fail to take into account many overheads during actual processing. However, it is a good indication of performance.

Concurrent input vectors: name given to a matrix of input vectors that are to be presented to a network "simultaneously." All the vectors in the matrix will be used in making just one set of changes in the weights and biases.

Conjugate gradient algorithm: in the conjugate gradient algorithms a search is performed along conjugate directions, which produces generally faster convergence than a search along the steepest descent directions.

Continuous output: An output neuron that can have any value. For example in a control system an output neuron controlling the speed of a motor would have to have a continuous output (cf. Binary output).

Connectionism: This term is based on an assumption shared by most massively parallel computational formalisms: that only a small number of bits of information can be sent from one processor to another. Hence, an important

conventional computer mechanism: i.e., passing complex symbolic structures - cannot be used directly. So the burden of computation is put on the connection structure of the network. 'Connectionist' systems have become largely synonymous with neural networks [Demuth 1996].

Connectivity: Neural networks exhibit several kinds of patterns of connectivity between their processing elements depending on the neural network model being used. Processing elements or nodes may be fully connected locally connected to neighbouring nodes or sparsely connected to a few distant nodes. In addition networks may be layered and the processing elements or nodes in these layers linked by means of feedback or feed-forward connections.

See also "Feedback", "Feed-forward", "Full Connectivity", "Local Connectivity", "Nearest Neighbour Connectivity", "Sparse Connectivity" and "Neural Network".

Conventional computer: A computer that is programmed by sequential instructions as to precisely how to process inputs to produce outputs. Conventional computers must follow fixed, predefined programmes or rules.

Convergence: A neural network is said to have converged when the training error has reached a pre-set threshold, which indicates the network has successfully been taught.

Cross talk: The overlap of input patterns in a neural network, which can result when a network does not have enough processing elements to allow one element or a group of elements to be reserved exclusively for every possible input pattern.

Cycle: a single presentation of an input vector, calculation of output and new weights and biases.

Data fitting: Conventional statistics try to fit input data to a fixed equation. This is often slow and can miss important features. It also tends to smooth out the data. If the training set for a neural network is too small this can lead to a similar result.

Darwin III Automaton: A type of neural networks model using self-supervised training; this model is a complex simulated automaton that learns to follow a moving target and touch the target with a multi-jointed arm. It is an instantiation of a developing theory of brain function called "neural Darwinism".

Decision boundary: a line, determined by the weight and bias vectors, for which the net input *n* is zero.

Decision network: The output from a decision network is usually a single value, which indicates the best choice for the given input data.

Dead neurons: a competitive layer neuron that never won any competition during training and so has not become a useful feature detector. Dead neurons do not respond to any of the training vectors.

Delta rule: the Widrow-Hoff rule.

Delta vector: the delta vector for a layer is the derivative of a network's output error with respect to that layer's net input vector.

Distance: the distance between neurons, calculated from their positions with a distance function.

Distributed Representation: Each entity or concept is represented by a pattern of activity distributed over many processing elements and each processing element is involved in representing many different entities or concepts. As opposed to local or unary representation. See also "Grandmother Cells" and "Local Representation" [Darpa , Haykin 1996].

Early stopping: a technique based on dividing the data into three subsets. The first subset is the training set used for computing the gradient and updating the network weights and biases. The second subset is the validation set. When the validation error increases for a specified number of iterations, the training is stopped, and the weights and biases at the minimum of the validation error are returned. The third set is the test set. It is used to verify the network design.

Epoch: the presentation of the set of training (input and/or target) vectors to a network and the calculation of new weights and biases. Note that training vectors may be presented one at a time or all together in a batch. (In short i.e., One complete presentation of the training set to the network during training).

Error jumping: a sudden increase in a network's sum-squared error during training. This is often due to too large a learning rate.

Error ratio: a training parameter used with adaptive learning rate and momentum training of BP networks.

Error vector: the difference between a network's output vector in response to an input vector and an associated target output vector.

Error: The difference between the network response indicated in the training set and the calculated network response during training.

Facts: (See training set)

Fault tolerance: The ability of a network to operate correctly with noisy data or when part of the network is damaged or missing.

Features: Within any group of patterns there may be some related data, which represent features.

Feature detector: A group of neurons within a network that are trained to recognise a feature.

Excitation: See "Neurotransmitters'" and "Weight".

Fan-in: The number of processing elements that either excite or inhibit a given unit.

Fan-out: The number of processing elements directly excited or inhibited by a given unit.

Feedforward network: a layered network in which each layer only receives inputs from previous layers.

Fletcher-Reeves update : a method developed by Fletcher and Reeves for computing a set of conjugate directions. These directions are used as search directions as part of a conjugate gradient optimization procedure.

Feedback network: Characterised by multi-layer neural networks with recursive connections that iterate over many cycles to produce an output. An example of a feedback neural network is the Hopfield Network. Contrasted with 'feed-forward'. See also "Connectivity" and "Feed-forward".

Feed-forward network: Characterised by multi-layer neural networks whose connections exclusively feed inputs from lower to higher layers; in contrast to a feedback network a feed-forward network operates only until its inputs propagate to its output layer. An example of a feed-forward neural network is the multi-layer perceptron. It is stable because there is no feedback, the network will produce a result in a single operation. (See also "Connectivity" and "Feedback")

Fixed Weight: See "Weight".

Function approximation: the task performed by a network trained to respond to input with an approximation of a desired function.

Full Connectivity: All processing elements or nodes in a neural network are connected to all other processing elements or nodes; also 'fully connected'. In contrast to local and sparse connectivity. See also "Connectivity", "Local Connectivity", "Nearest Neighbour Connectivity" and "Sparse Connectivity".

Generalisation: The ability of a network to produce a result (a prediction or guess) uses data on which it has not been trained.

Generalised predictive control: belongs to the class of long range predictive control [see appendix below]

Generalized regression network: approximates a continuous function to an arbitrary accuracy, given a sufficient number of hidden neurons.

Global minimum: the lowest value of a function over the entire range of its input parameters. Gradient descent methods adjust weights and biases in order to find the global minimum of error for a network.

Gradient descent: the process of making changes to weights and biases, where the changes are proportional to the derivatives of network error with respect to those weights and biases. This is done to minimize network error.

Golden section search: a linear search, which does not require the calculation of the slope. The interval containing the minimum of the performance is subdivided at each iteration of the search, and one subdivision is eliminated at each iteration.

Graceful Degradation: In neural networks the notion that no single processing element or neuron is essential to the network's operation; network performance gradually deteriorates as more and more processing elements are destroyed but there is no single critical point at which performance breaks down.

Hard limit transfer function: a transfer that maps inputs greater-than or equal-to 0 to 1, and all other values to 0.

Hebb learning rule: historically the first proposed learning rule for neurons. Weights are adjusted proportional to the product of the outputs of pre- and post-weight neurons. (Or alternatively: A learning algorithm in which the repeated excitation of the interconnection between two processing elements causes the strength or weight of that interconnection to increase.)

Home neuron: a neuron at the centre of a neighbourhood.

Hybrid bisection-cubicsearch - a line search that combines bisection and cubic interpolation.

Hamming Network: A neural network algorithm based on the Hopfield Network, which is used in pattern classification problems. The feed-forward Hamming Network is notable for requiring fewer connections than the Hopfield Network. Also called the 'unary Model'.

Hidden Units /layers: Those processing elements in multi-layer neural network architectures which are neither the input layer nor the output layer but are located in between these and allow the network to undertake more complex problem-solving (i.e., non-linear mapping) than networks with no hidden units. Also called 'hidden layers of processing elements'.

Hopfield Network: A type of neural network model characterised by full connectivity, feedback, and unsupervised training, which is, used in pattern classification and optimization problems.

Input layer: a layer of neurons receiving inputs directly from outside the network.

Initialization: the process of setting the network weights and biases to their original values.

Input space: the range of all possible input vectors.

Input vector: a vector presented to the network.

Input weights: the weights connecting network inputs to layers.

Input weight vector: the row vector of weights going to a neuron.

Jacobian matrix: contains the first derivatives of the network errors with respect to the weights and biases.

Jog weights: To submit all, or some, of the weights to a random influence. See also local minimum.

Kohonen: Inventor of the self-organising map (Professor T Kohonen)

Indium Bump Bonding: A technology under development for connecting non-silicon infrared detector arrays to silicon integrated circuit preamplifiers and signal processors. Bump bonding is being proposed as a means of achieving massively parallel interconnections of neural networks.

Inhibition: See "Neurotransmitters" and "Weight".

Input: See "Processing Element" and "Weight".

Interconnect: The links or information channels between a neural network's processing elements. The pattern of these interconnections must be appropriate to the neural network's application. See also "Connectivity", "Neural Network", "Processing Element" and "Weight".

Kohonen Self-organizing Feature Map: A type of neural network learning algorithm which does not require explicit tutoring of input-output correlation's and spontaneously self-organizes upon presentation of input information patterns. It is used in optimization and pattern classification problems. (Inventor of the self-organising map (Prof. T Kohonen))

Layer diagram: a network architecture figure showing the layers and the weight matrices connecting them. Each layer's transfer function is indicated with a symbol. Sizes of input, output, bias and weight matrices are shown. Individual neurons and connections are not shown.

Layer weights: the weights connecting layers to other layers. Such weights need to have non-zero delays if they form a recurrent connection (i.e. a loop).

Learning: the process by which weights and biases are adjusted to achieve some desired network behaviour. (See training)

Learning rate: a training parameter that controls the size of weight and bias changes during learning.

Learning rules: a procedure for modifying the weights and biases of a network.

Learning Algorithms In neural networks the equations which modify some of the weights of processing elements in response to input signals and values supplied by the transfer function; the learning algorithm(s) employed in a neural network allow the elements responses to input signals to change over time.

Learning rule: The algorithm used for modifying the connection strengths, or weights, in response to training patterns while training is being carried out.

Learning mode: Where the neural network is being trained. See Training.

LMS (Least Mean Square) Algorithm: A modification to the perceptron convergence procedure, which can form the least mean, squared solution in certain problems. This solution, used the Adaline, minimizes the mean squared error between the desired output of a perceptron-like network and the actual output. See also "Adaline".

Local Connectivity: The processing elements or nodes, in one layer of a multi-layer neural network are connected only to the corresponding nodes in other layers. In contrast full and sparse connectivity. See also "Connectivity", "Full Connectivity", "Nearest Neighbour Connectivity" and "Sparse Connectivity".

Levenberg-Marquardt: an algorithm that trains a neural network 10 to 100 faster than the usual gradient descent BP method. It will always compute the approximate Hessian matrix, which has dimensions.

Line search function: procedure for searching along a given search direction (line) to locate the minimum of the network performance.

Linear transfer function: a transfer function that produces its input as its output.

Link distance: the number of links, or steps, that must be taken to get to the neuron under consideration.

Local minimum: The error is reduced during training to below a pre-set threshold. However, if a local minimum is encountered then the network may never train successfully. The training algorithm tries to reduce the error but is prevented from doing so because the current error is only a local minimum. It is difficult to determine if a minimum it local or global (q.v.) and the usual method to get out of a local minimum is to jog (q.v.) the weights and continue training.

Local Representation: In neural networks, the use of one processing element to represent each entity or concept. Also called unary representation; as opposed to distributed representation. See also "Distributed

Representation" and "Grandmother Cells".

Log-sigmoid transfer function: a squashing function of the form shown below that maps the input to the interval (0,1). (The toolbox function is logsig): $f(n) = \frac{1}{1+e^{-n}}$

Mapping: Transformation of data from one representation to another. For example a Cartesian co-ordinates pattern might be translated into polar co-ordinates.

Markov Random Field Network: A type of neural network algorithm used in optimisation problems and closely related to cellular automata.

Maximum performance increase: the maximum amount by which the performance is allowed to increase in one iteration of the variable learning rate-training algorithm.

Maximum step size: the maximum step size allowed during a linear search. The magnitude of the weight vector is not allowed to increase by more than this maximum step size in one iteration of a training algorithm.

Mean square error function: the performance function that calculates the average squared error between the network outputs ***a*** and the target outputs *t*.

Minimum variance Tracking: This strategy concerns the design of a controller ensuring a minimum variance of the control variable (plant output) around the reference, in the presence of random disturbances [Landau 1998]

Momentum: a technique often used to make it less likely for a BP networks to get caught in a shallow minima.

Momentum constant: A training parameter that controls how much "momentum" is used.

Multi-layer Perceptron: A multi-layer feedforward neural network that is fully connected and which is typically trained by the back-propagation learning algorithm.

Neighbourhood: a group of neurons within a specified distance of a particular neuron. The neighbourhood is specified by the indices for all of the neurons that lie within a radius of the winning neuron i^*: $N_i(d) = \{j, d_{ij} \leq d\}$.

Nearest Neighbour Connectivity: A type of local connectivity in which a neural network's processing elements or nodes are connected to those processing elements or nodes which are physically contiguous. See also "Connectivity", "Full Connectivity", "Local Connectivity" and "Sparse Connectivity".

Net input vector: the combination, in a layer, of all the layer's weighted input vectors with its bias.

Neuron: the basic processing element of a neural network. Includes weights and bias, a summing junction and an output transfer function. Artificial neurons, such as those simulated and trained with this toolbox, are abstractions of biological neurons.

Neuron diagram: a network architecture figure showing the neurons and the weights connecting them. Each neuron's transfer function is indicated with a symbol.

Ordering phase: period of training during which neuron weights are expected to order themselves in the input space consistent with the associated neuron positions.

Neocognitron: A type of neural network model used in pattern classification problems. The neocognitron model combines an unsupervised learning algorithm with a multi-layer architecture designed to provide pattern recognition with tolerance to positional shifts, geometric distortion and scale variation.

Neural computing: the implementation on a computer of a neural network.

Neural Network: An information processing system, which operates on inputs to extract information and produces outputs corresponding to the extracted information. Also called "artificial neural networks", "connectionist models", "parallel distributed processing models", and "neuromorphic systems" [Darpa].

Specifically, a neural network is a system composed of many simple processors fully, locally or sparsely connected - whose function is determined by the connection topology and strengths. This system is capable of a high-level function, such as adaptation or learning with or without supervision, as well as lower-level functions, such as vision and speech pre-processing. The function of the simple processor and the structure of the connections are inspired by biological nervous systems.

The key attributes of neural networks are

(a) massive parallelism, which results in high-speed decisions and potential fault tolerance and

(b) adaptivity, which means neural networks can be trained rather than programmed in the classical way, and their performance may improve with experience.

A neural network is described by either an algorithm (which specifies the functional transformation from inputs to outputs) and/or an implementation (the physical realisation of the processing mechanism that runs the algorithm). See also "Connectivity", "Processing Element" and "Weight".

Neuro-dynamics : The study of the generation and propagation of synchronised neural activity in biological systems.

Neuron: A single processing element within a neural network. A neuron receives its input from the outputs of other neurons or signals from the outside world. The neuron uses this information to produce its output using a simple mathematical formula. This output is then fed to other neurons or directly to the outside world. There are many different types of neurons. A neuron is also a nerve cell within the brain.

Neurotransmitters: In biological systems, specialised molecules that act across synapses and which open up neural membrane channels that permit ionic currents (i.e., action potentials) to act. Neurotransmitters and currents either depolarize the membrane, resulting in neural excitation or hyperpolarize it, resulting in neural inhibition. Some 50 different neurotransmitters have been identified so far; some appear to play an important role in determining patterns of neural interconnections. See also "Axon", "Dendrite", "Neuron", "Soma" and "Synapse" [Darpa].

Optimising network: Some neural network architectures are good at solving problems which require picking the best combination from a large number of possible combinations (e.g. what is the best route so that one visits 10 customers in minimum time'). Conventional approaches take a long time to arrive at a solution; neural networks can solve the problem in a shorter time.

Output layer: a layer whose output is passed to the world outside the network.

Output neuron: A neuron within a neural networks whose outputs is the result of the network.

(Also see "Processing Element" and "Weight".)

Output vector: the output of a neural network. Each element of the output vector is the output of a neuron.

Output weight vector: the column vector of weights coming from a neuron or input. (See outstar learning rule.)

Outstar learning rule: a learning rule that trains an neuron's (or input's) output weight vector to take on the values of the current output vector of the post-weight layer. Changes in the weights are proportional to the neuron's output.

Overfitting: a case in which the error on the training set is driven to a very small value, but when new data is presented to the network, the error is large.

Over training: Training in neural networks involves feeding into the network facts about the problem to be solved. If the network is trained only on this sequence of facts then it becomes less able to generalise. Over training is the result of training the neural network to respond very accurately to the training set only. The effects of over training can be reduced by using a test set during the training process.

Paradigm: (See training)

Pattern association: the task performed by a network trained to respond with the correct output vector for each presented input vector.

Pattern recognition: the task performed by a network trained to respond when an input vector close to a learned vector is presented. The network "recognizes" the input as one of the original target vectors.

Performance function: commonly the mean squared error of the network outputs. However, the toolbox also considers other performance functions. Type nnets and look under performance functions.

Perceptron: a single-layer network with a hard limit transfer function. This network is often trained with the perceptron learning rule. See also "Adaline", "LMS Algorithm", "Multi-layer Perceptron" and "Single-layer Perceptron".)

Perceptron learning rule: a learning rule for training single-layer hard limit networks. It is guaranteed to result in a perfectly functioning network in finite time given that the network is capable of doing so.

Positive linear transfer function: a transfer function that produces an output of zero for negative inputs and an output equal to the input for positive inputs.

Postprocessing: converts normalized outputs back into the same units, which were used for the original targets.

Powell-Beale restarts: a method developed by Powell and Beale for computing a set of conjugate directions. These directions are used as search directions as part of a conjugate gradient optimization procedure. This procedure also periodically resets the search direction to the negative of the gradient.

Preprocessing: Modification of the data before they are applied to the input layer of the neural network. For example; scaling, FFT, filtering, averaging, etc.)

Post-processing: Modification the results of the neural network before being applied to the real world For example; scaling, filtering, etc.

Principal component analysis: orthogonalize the components of network input vectors. This procedure can also reduce the dimension of the input vectors by eliminating redundant components.

Parallel processing A computing technique that carries out multiple tasks simultaneously. Neural computing is ideally suited to parallel processing because a neural network acts upon all the information provided at the same time. However, most neural computers use conventional sequential processing due to cost. (Also see Processing Element)

Prediction network: A network that produces real-valued outputs as a response to the data fed into it. For example, a prediction network could be used to predict the stock market results for the following day.

Pattern Classifiers: Mappings that define partitioning of feature space into regions corresponding to class membership.

Processing Element: The simple processors (also called 'neurons' after their biological inspiration or simply 'units') that are the essential units of which a neural network is comprised. Every processing element, which is endowed with only a small amount of local memory, receives one or more inputs from other processing elements or from external sources; these inputs are then modified by some weighted value specific to each input according to a learning algorithm. The sum of the products of the different weight times their individual inputs is then computed by the processing element. The processing element generates a single output signal that depends on these input sums. This single output signal can be fanned out to some number of other processing elements or be used as output from the network.

See also "Connectivity", "Neural Network", neuron and "Weight".

Quasi-Newton algorithm: class of optimization algorithm based on Newton's method. An approximate Hessian matrix is computed at each iteration of the algorithm based on the gradients.

Radial basis networks: a neural network that can be designed directly by fitting special response elements where they will do the most good.

Radial basis transfer function: the transfer function for a radial basis neuron is: $radbas(n) = e^{-n^2}$

Regularization: involves modifying the performance function, which is normally chosen to be the sum of squares of the network errors on the training set, by adding some fraction of the squares of the network weights.

Resilient BP: a training algorithm that eliminates the harmful effect of having a small slope at the extreme ends of the sigmoid "squashing" transfer functions.

Receptive Fields: In some multi-layer neural networks a processing element in the hidden layer(s) may receive input from a group of neighbouring units called the receptive field.

Reduced Coulomb Energy (RCE) Network: A type of neural network model used in general classification problems and characterized by a sparse, feed-forward connection topology and supervised training.

Relaxation: In neural networks, the notion that computation proceeds by iteratively seeking to satisfy a large number of weak restraints; thus connections represent constraints on the co-occurrence of pairs of processing elements. The network settles into a solution rather than calculating one.

Run mode: When the neural network is being executed.

Saturating linear transfer function: a function that is linear in the interval (-1, +1) and saturates outside this interval to -1 or +1. (The toolbox function is satlin.)

Scaled conjugate gradient algorithm: avoids the time consuming line search of the standard conjugate gradient algorithm.

Sequential input vectors: a set of vectors that are to be presented to a network "one after the other." The network weights and biases are adjusted on the presentation of each input vector.

Self-organization (map) (SOM): The autonomous modification of the dynamics of a complete neural network via learning in some or all of its processing elements to achieve a specified result. See also "Self-supervised Training", "Supervised Training" and "Unsupervised Training".

Self-supervised Training: A means of training adaptive neural networks; self-supervision is used by automata which require internal error feedback to perform some specific task. For example, automata which learn to track a moving spot by controlling simulated eye muscles can generate an error signal based on the distance between the position of the spot on a simulated retina and the centre or fovea of the retina. See also "Self-organisation", "Supervised Training" and "Unsupervised Training".

Serial processing: In which actions are processed sequentially.

Sigma parameter: determines the change in weight for the calculation of the approximate Hessian matrix in the scaled conjugate gradient algorithm.

Sigmoid: monotonic S-shaped function mapping numbers in the interval ($(-\infty, \infty)$) to a finite interval such as (-1, +1) or (0,1).

Simulation: takes the network input *p*, and the network object *net*, and returns the network outputs *a*.

Simulation software: Software packages that permit the training and validation of a neural network prior to implementing it.

Simulated Annealing: A stochastic computational technique derived from statistical mechanics for finding near globally minimum cost solutions to large optimisation problems.

Single-layer Perceptron: A type of neural network algorithm used in pattern classification problems and trained with supervision. The single-layer perceptron generated much interest when it was initially developed in the 1950s by Rosenblatt because of its ability to learn to recognize simple patterns. Connection weights and the thresholds in a perceptron can be fixed or adapted using a number of different algorithms. See also "Adaline" and "Perceptron".

Site Function: In a neural network, a processing element's inputs are connected to specific sites. A processing element may have more than one "input site". Each site has an associated site function which carries out local computation based on the input values at the site.

Soma: In biological systems, the large, round central body of a neuron which contains the genetic and metabolic machinery necessary to keep the neuron alive. See also "Axon", "Dendrite", "Neuron" "Neurotransmitters" and "Synapse".

Sparse Connectivity: The processing elements or nodes in a neural network are connected to only a few distant other processing elements, or nodes. In contrast to full and local connectivity. See also "Connectivity", "Full Connectivity", "Local Connectivity" and "Nearest Neighbour Connectivity".

Spread constant: the distance an input vector must be from a neuron's weight vector to produce an output of 0.5.

Squashing function: a monotonic increasing function that takes input values between and and returns values in a finite interval.

Star learning rule: a learning rule that trains a neuron's weight vector to take on the values of the current input vector. Changes in the weights are proportional to the neuron's output.

Stochastic: A process involving a randomly determined sequence of observations, each of which is considered as a sample of one element from a probability distribution. Stochastic variation implies randomness as opposed to a fixed rule or relation in passing from one observation to the next in order.

Sum-squared error: The sum of squared differences between the network targets and actual outputs for a given input vector or set of vectors.

Synapse: In biological systems, the tissues connecting neurons. Synapses are the specialised contacts on a neuron, which are the termination point for axons from other neurons. Synapses make contact with the dendrites from other neurons and are capable of changing a dendrite's local potential in a positive or negative direction. See also "Axon", "Dendrite", "Neuron", "Neurotransmitters" and "Soma" [Darpa].

supervised learning: A learning process in which changes in a network's weights and biases are due to the intervention of any external teacher. The teacher typically provides output targets.

Supervised training: A means of training adaptive neural networks, which requires labelled training data and an external teacher. The teacher knows the desired correct response and provides an error signal when an error is made by the network. This is sometimes called 'reinforcement learning' or 'learning with a critic' when the teacher only indicates whether a response was correct or incorrect but does not provide detailed error information. Also called "hetero-associative learning". As opposed to unsupervised training, self-organization and auto-associative learning. See also "Self-organization", "Self-supervised training" and "Unsupervised Training".

Symmetric hard limit transfer function: a transfer that maps inputs greater-than or equal-to 0 to +1, and all other values to -1.

Symmetric saturating linear transfer function: produces the input as its output as long as the input i in the range -1 to 1. Outside that range the output is -1 and +1 respectively.

Tan-sigmoid transfer function: a squashing function of the form shown below that maps the input to the interval (-1,1). (The toolbox function is tansig.): $f(n) = \frac{1}{1+e^{-n}}$

Tapped delay line: a sequential set of delays with outputs available at each delay output.

Target vector: the desired output vector for a given input vector.

Test set threshold: The test set comprises information about the problem to be solved that the network has not previously seen. Once training has been completed the network should be validated using a test set.

Topology: The way in which the neurons are connected together determines the topology of the neural network. Sometimes referred to as architecture or paradigm. Some examples are self-organising map, multi-layer perceptron.

Topology functions: ways to arrange the neurons in a grid, box, hexagonal, or random topology.

Training: Training is the process by which the neural network connection weights are adjusted so that the network performs the function for which it is designed.)

Training vector: an input and/or target vector used to train a network.

Transfer function: the function that maps a neuron's (or layer's) net output ***n*** to its actual output.

Tuning phase: period of SOFM training during which weights are expected to spread out relatively evenly over the input space while retaining their topological order found during the ordering phase.

Training set: A neural network is trained using a training set. A training set comprises information about the problem to be solved as input stimuli. In some computing systems the training set is called the facts file.

Transfer Function: The differential (or difference) equations, which determine each, processing element's operation. These equations describe how the output signal evolves in time as a function of the input signals.

Threshold: A function f with values 1 and -1 is called a *threshold function* if the inverse image of 1, $f^{I}(1)$, and of -1, $f^{I}(-1)$, are separated by a hyperplane [Wilde 1997].

Turing Machine: (Cambridge mathematician (1912-1953), who described (1936) a theoretical ' Turing Machine' whose abstract structure must be common to all possible numerical computers. He also considered the question ' Can machine think' and proposed a critical test for comparing human thought with mechanical results [John and Litter 1984].

Underdetermined system: a system that has more variables than constraints.

Unsupervised learning: A means of training adaptive neural networks, which requires unlabeled training data and no external teacher. Data is presented to the network and internal categories or clusters are formed which compress the amount of input data that must be processed at higher levels without losing important information. This clustering task is sometimes called 'vector quantization'. See also "Self-organization", "Self-supervised Training" and "Supervised Training".)

Update: make a change in weights and biases. The update can occur after presentation of a single input vector or after accumulating changes over several input vectors.

Validation set: (See Test set)

Value Unit Coding: In neural network data representation, the encoding of the value of a variable as the location of an active processing element, or node. Such nodes are often arranged in an orderly fashion to form a topographic map of some external variable.

Variable Unit Coding: In neural network data representation, the encoding of the value of a variable as the amplitude of the output of a processing element or node.

Vector Quntization: See "Unsupervised training".

Viterbi Network: A neural network architecture, which implements a temporal, decoding algorithm used for non-linear analogue processing based speech recognition.

Weight: The value associated with a connection between neurons in a neural network. This value determines how much of the output of one neuron is fed to the input of another.

Weighted input vector: the result of applying a weight to a layer's input, whether it is a network input or the output of another layer.

Weight matrix: a matrix containing connection strengths from a layer's inputs to its neurons. The element $w_{i,j}$ of a weight matrix W refers to the connection strength from input j to neuron i.

Widrow-Hoff learning rule: a learning rule used to trained single layer linear networks. This rule is the predecessor of the BP rule and is sometimes referred to as the delta rule.

SPECIFIC DEFINITION

A direct control system design. "Direct" means that the controller is a neural network. A neural network controller is often advantageous when the real-time platform available prohibits complicated solutions. The implementation is simple while the design and tuning are difficult implying a retraining of the network every time a design parameter is modified. Often this training has to be performed according to an on-line scheme.

An indirect control system design. This class of designs is always model based. The idea is to use a neural network to model the system to be controlled. This model is, then employed in a more "conventional" controller design. The model is typically trained in advance, but the controller is designed on-line. As it will appear, the indirect design is very flexible; thus, it is the most appropriate for the majority of common control problems.

REFERENCES

[1] Johan Glenn and Graham Litter (Editors). A Dictionary of Mathematics. Copyright © 1984 J.A. Glenn and G.H.Littler.

[2] Matlab Neural Network Tool box User Guides.

[3] DARPA Neural Network Study. October 1987- February 1988. Reprinted March 1990, 1992. Published by AFCEA International Press.

[4] Hagan Demuth Beale. Neural Network Design. Copyright © 1996 by PWS Publishing Company. (Also see list of main references).

APPENDIX B: ABSTRACTS OF PUBLISHED PAPERS

Back-propagation Neural Network Enhanced Non-linear System Identification and Control.

B. K. Thapa and T. Earthrowl-Gould.

Aston University, School of Engineering and Applied Science, Birmingham B4 7ET (UK)

ABSTRACT

Neural Networks (NN) are the focus of much interest within the control engineering community because they offer a new analytical approach to solve many complex applications which are often problematic when a traditional analytical methods are adopted. This paper explores the use of back-propagation (BP) algorithms in NN learning in the context of applying these solutions to recognisable system identification such as modelling, predictive, self-tuning and direct NN control applications. BP algorithms are shown to be applicable to static and dynamic, single and multi-layer as well as recurrent NN architectures. The architectures and algorithms described here are indicative of on going research on complex non-linear structures of neural controllers and as such provide an overview of the state of the art.

Simulations examples are presented to demonstrate the algorithm presented in this paper and results indicate that the identification and self-tuning scheme can possibly deal with a complex unknown non-linearity.

Non-linear Control with Neural Networks

B. K. Thapa, B.Jones and Q.M.Zhu

Aston University, School of Engineering and Applied Science, Birmingham, B4 7ET (UK)

ABSTRACT

This paper is concerned with a non-linear self-tuning tracking problem using back-propagation (BP) neural learning and system identification techniques. Traditional self-tuning adaptive control techniques can only deal with linear systems or special non-linear systems. BP neural networks have the capability to learn arbitrary non-linearities and show great potential for adaptive control applications. A scheme for combining BP neural networks with self-tuning adaptive control techniques is proposed. Two simple simulation studies are provided to illustrate the effectiveness of the control algorithm. Simulation indicates that the indentification self-tuning scheme can possibly deal with a complex unknown non-linearity.

APPENDIX C

MATLAB, SIMULINK, Toolboxes and S-Functions are for modelling, simulation and implementation. The objective of this chapter is to highlight and describe the software package used for this research work.

C1: WHAT IS MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation, Algorithm development, Modelling, simulation and prototyping
- Data analysis, exploration and visualisation, Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical-computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The name MATLAB stands for ***matrix laboratory***. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation [1-4].

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis [4].

MATLAB features a family of application-specific solutions called *toolboxes*. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialised technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others [4].

C2: WHAT IS SIMULINK?

SIMULINK is a software package for modelling, simulating and analyzing dynamic systems. It supports linear and non-linear systems, modelled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate, i.e., have different parts that are sampled or updated at different rates [2].

As an optional extension to the core of MATLAB, SIMULINK provides a graphical user interface (GUI) for constructing block diagram models of dynamic systems. Large libraries of building blocks are provided such as sources, linear and nonlnear components and connectors. This makes it possible to create your own blocks and model a system rapidly, clearly, and without having to write a single line of simulation code. And, because the models you create are graphical in nature, SIMULINK gives you simulation, documentation and publication-quality output - all from the same screen.

Models are hierarchical, so you can build models using both top-down and bottom-up approaches. You can view the system at a high-level, then double-click on blocks to go down through the levels to see increasing levels of model detail. This approach provides insight into how a model is organized and how its parts interact see reference for detail [1].

After you define a model, you can simulate it, using a choice of integration methods, either from the Simulink menus or by entering commands in MATLAB's command window. The menus are particularly convenient for interactive work, while the command-line approach is very useful for running a batch of simulations (for example, if you are doing Monte Carlo simulations or want to sweep a parameter across a range of values). Using scopes and other display blocks, you can see the simulation results while the simulation is running. In addition, you can change parameters and immediately see what happens, for "what if" exploration. The simulation results can be put in the MATLAB workspace for post processing and visualisation. See reference [1] for extra details.

C3: WRITING S-FUNCTIONS AND OVERVIEW OF S-FUNCTIONS

What Is an S-Function?
When to Use an S-Function
How S-Functions Work
Overview of M-File and C MEX S-Functions
S-Function Concepts

C3_1: Introduction

S-functions (system-functions) provide a powerful mechanism for extending the capabilities of Simulink [1]. S-functions allow you to add your own algorithms to Simulink models. You can write your algorithms in MATLAB or C by following a set of simple rules; you can implement your algorithms in an S-function. After you have written your S-function and placed its name in an S-Function block (available in the Non-linear Block sub-library), you can customize the user interface by using masking. You can also customize the code generated by the Real Time Workshop for S-functions by writing a Target Language Compiler TM (TLC) [2].

C3_2: What Is an S-Functions?

An *S-function* is a computer language description of a dynamic system. S-functions can be written using MATLAB or C. C language S-functions are compiled as MEX-files using the mex utility described in the *Application Program Interface Guid*e [2]. As with other MEX-files, they are dynamically linked into MATLAB when needed.

S-functions use a special calling syntax that enables you to interact with Simulink's equation solvers. This interaction is very similar to the interaction that takes place between the solvers and built-in Simulink blocks.

The form of an S-function is very general and can accommodate continuous, discrete, and hybrid systems. As a result, nearly all Simulink models can be described as S-functions. S-functions are incorporated into Simulink models by using the S-Function block in the Non-linear Block sub-library.

C3_3: When to Use an S-Function

The most common use of S-functions is to create custom Simulink blocks. You can use S-functions for a variety of applications, including:

- Adding new general purpose blocks to Simulink
- Incorporating existing C code into a simulation
- Describing a system as a mathematical set of equations
- Using graphical animations or simulation (e.g. see simulations results as presented in the thesis or see any demo within Matlab e.g. the inverted pendulum demo etc.)

An advantage of using S-functions is that you can build a general-purpose block that you can use many times in a model, varying parameters with each instance of the block.

C3_4: How S-Functions Work

Each block within a Simulink model has the following general characteristics: a vector of inputs, u, a vector of outputs, y, and a vector of states, x, details can be found in User guide [1]

C4: WHAT IS TOOLBOXES?

Application-Specific end Extensible

MATLAB features a family of application-specific products called Toolboxes. Toolboxes are comprehensive libraries of MATLAB functions that customise the MATLAB environment for particular classes or problems and application areas. Here only relevant tooboxes are described which is directly linked with this research work.

The MATIAB application toolboxes represent the work of some of the worlds top researchers in fields such as signal processing, automatic control, and neural networks, Toolboxes let you "stand on the shoulders" of world-class scientists and researchers who are defining the state of the art and implementing it in MATLAB.

Toolboxes combine the advantages of pre-packaged "off-the-shelf " software with the inherent power and flexibility of a technical computing environment [4]:

- Toolbox is built on top of MATLAB's fast and reliable numeric.
- A full complement of graphics and visualisation tools are always available to examine results.
- MALAB's open system approach gives you access to toolbox source code, so that you can inspect, customise, and extend the algorithms and functionality of the toolbox to suit your needs.
- All toolboxes are available on the wide variety of computer platforms on which MATLAB runs.
- Because they all share a common basis in MATLAB, toolboxes can be used together in a seamless manner. For example, you can apply optimisation and neural network tools to advanced signal processing problems and display the results as colour 3-D graphs -- all in a single environment!

A large family of toolbox products are available from The MathWorks, the highlights of which are listed on the following pages.

Control System Design: Automatic control system design and analysis tools. Classical and modern techniques; Continuous- and discrete-time; State-space and transfer function models; System interconnection; Transformation between models; Model reduction; Frequency response: Bode, Nyquist, Nichols, SVD; Time response: impulse, step, ramp, general; Root-locus, pole-placement, LQR, LQG

Fuzzy Logic Toolbox: An intuitive graphical environment for designing with fuzzy logic systems based theory

Image Processing: 2-D filter design and filtering; Image restoration and enhancement; Colour, geometric, and morphological operations; Two-dimensional transforms; Image analysis and statistics

Non-linear Control Design Toolbox: A revolutionary, interactive approach to computer-aided control system design

Optimization Toolbox: Tools for the general optimization of linear and non-linear functions

Neural Networks: Neural network design and simulation tools provide an environment for developing neural networks within MATLAB. Up to date known functions are as follows:

- Associative, backpropagation, feature map, Hopfield, Kohonen, self-organising, Widrow-Hoff networks; Competitive, limit, linear, sigmoid transfer functions; Feedforward, Recurrent architectures; Performance analysis functions and graphs; Unlimited layers elements, interconnections.

System Identification: Advanced signal processing tools for parametric modelling, system identification, and time-series analysis.

- Provides a flexible graphical user interface that aids in the building of accurate, simplified models of complex system from noisy time-series data.

Real-Time Workshop: Generates fast, target-independent C code from Simulink block diagrams.

Robust Control: Leading-edge robust-control system synthesis tools to deal with systems in the presence of uncertainty.

- LQG/LTR optimal control synthesis; H_2 and $H\infty$ optimal control synthesis ; Singular-value model reduction; Spectral factorisation and model building

Signal Processing Toolbox: Powerful tools for algorithm development, signal and linear systems models, in addition to specialized GUIs for filter design and spectral analysis.

- Digital and analogue filter design and implementation; Spectrum analysis and estimation; Filter response simulation; FFT, DCT, and other transforms; Parametric modelling; Multi-rate signal processing ; Modulation and demodulation

For further information of other toolboxes can be found Mathworks [4].

C5: OPEN AND EXTENSIBEL ARCHITECTURE

Extensible, Connectable, and interoperable

The open architecture allows you to extend the simulation environment (See Fig.1 below):

- Create custom blocks and block libraries with your own icons and user interfaces from MATLAB, Fortran, or C code,
- Link in pre-existing Fortran and C simulation code to preserve your validated models.
- Generate C code from your models, using the optional SIMCLINK C Code Generator.

Cross-Platform Interoperability

- MATAB is available on industry standard computing platforms, ranging from personal computers and workstations to minicomputers and supercomputers.
- MATLAB is designed to operate in a multi-vendor heterogeneous network environment, enabling workgroups to share a common set of data and toolboxes, while using a diverse collection of machines.
- MATLAB supports the industry standard windowing systems: MS-Windows, X Windows, and Macintosh.
- User-generated applications and data are transferable across the entire range of MATLAB platforms without modification. Any needed conversions are performed automatically.
- Many MATLAB features and virtually all toolbox features, are implemented in programmable *"m-files"* that give you access to the source code and algorithms.
- MATLAB's pioneering open system approach enables you to inspect algorithms, make changes to existing functions and add your own new features see Fig. 1.

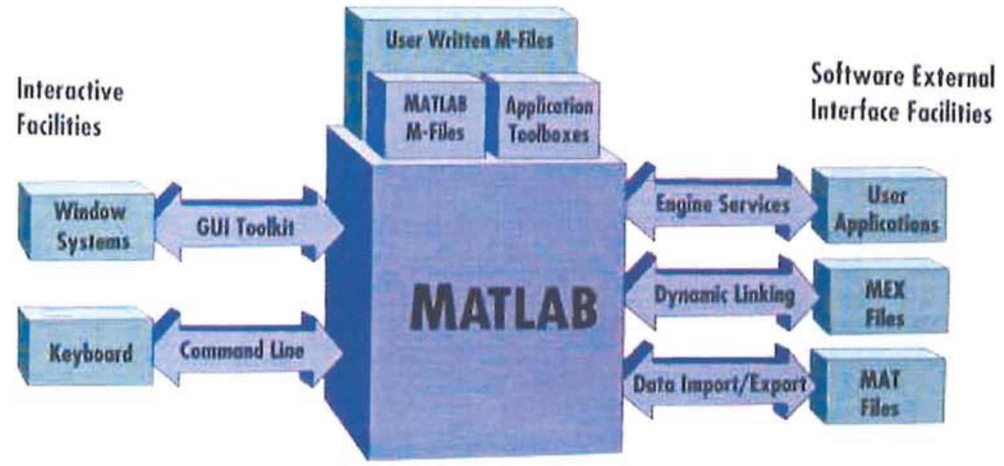


Fig.1 Open Architecture of MATLAB

C6: SUMMARY

MATLAB is design, analysis and data visualisation tools where as SIMULINK is system modelling, simulation and validation.

S-functions allow you to add custom code to your SIMULINK model. You can embed the code directly into the generated code or, by default, allow the S-function Application Program Interface (API) to call the S-function. Embedding the S-function code directly into the generated code is called *inlining* the S-function. The default is called *noninlined* [2]

MATLAB offers a unique simulation and prototyping environment. The powerful technical language is both concise and descriptive, allowing you to model complex systems with small sections of easy-to-follow code. The vast library of functions, available as part of the MATLAB code and through the application-oriented toolboxes, allows you to quickly build simulations and models for a variety of application types. Tools for modelling range from differential equation solvers in MATLAB, to specialized toolbox functions for statistics and neural network training and modelling. Built-in animation functions and fast graphics allow you to visualize model behaviour for analysis, testing and debugging, and presentation purposes.

Advantages

- Since MATLAB is an interpreted language, you can modify your models to see the effects immediately, without the additional overhead of recompilation as in C. Because so many of the low-level and advanced math algorithms are already developed for you, the code required to build a model in MATLAB is significantly shorter than the corresponding C or C++ code. This compactness makes MATLAB code easy to write and to maintain over time.
- A Set of Targeted Simulation Approaches where MATLAB offers a familiar programming environment.
- Simulink and State-flow provide a graphical, design environment for modelling and simulating complex control, DSP, and supervisory logic systems. Built on MATLAB, these products can call any MATLAB function including user-written routines, allowing you to combine the best of both approaches.
- Even toolbox functions can be embedded within Simulink block-diagram models. Simulation and Modelling Products
- MATLAB Compiler translates MATLAB code to ANSI standard C and C++ code

Disadvantage:

- It is expensive for licensing the product and not easily available like other packages such a C or C++.

References

[1] User Guide: SIMULINK®. © Copyright 1994-1999 by the MathWorks, Inc.

[2] User Guide: Real-Time Workshop® for Use with SIMULINK®. © Copyright 1994-1999 by the MathWorks, Inc.

[3] User Guide: Neural Network ®. © Copyright 1994-1999 by the MathWorks, Inc.

[4] http://www.mathworks.com

[5] Target Language Compiler Reference Guide and the Real-Time Workshop User's Guide© Copyright 1994-1999 by the MathWorks, Inc.

[6] MATLAB® User guides: Control Toolbox, Neural Network, Robust, Multivariable toolbox, Non-linear control toolbox, Optimisation toolbox, Signal Processing Toolbox, System Identification Toolbox.

APPENDIX D: DIGITAL CONTROL STRATEGIES

Design of digital controllers for SISO systems described by discrete time models in input-output form.

Comparing the control objectives in the time domain associated with the various control strategies one can classify these strategies in two categories:

1. *One step ahead predictive control.* In these strategies one computes a prediction of the output at *t+d+1* (*d* integer delay of the plant) namely $\hat{y}(t+d+1)$ as a function of $u(t), u(t-1),..., y(t), y(t-1),...$ and one computes *u(t)* such that a control objective in terms of $\hat{y}(t+d+1)$ be satisfied. This is illustrated in Fig.1.1

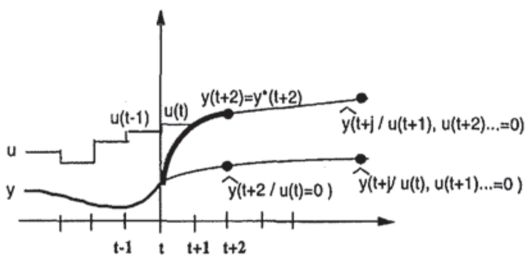


Fig.1.1: One step-ahead predictive control

2. *Long range predictive control.* In these strategies the control objective is expressed in terms of the future values of the output over a certain horizon and of a sequence of future control values. In order to solve the problem, one needs to compute:

$$\hat{y}(t+d+1), \hat{y}(t+d+2),..., \hat{y}(t+d+j)$$

which are expressed as:

$$\begin{aligned}\hat{y}(t+d+1) &= f_1(y(t), y(t-1),...,u(t),u(t-1),...)\\ &\vdots\\ \hat{y}(t+d+j) &= f_1(y(t), y(t-1),...,u(t),u(t-1),...)\\ &\quad g_j(u(t+1),...,u(t+j-1))\end{aligned}$$

To satisfy the control objective the sequence of present and future values the control $u(t), u(t+1),..., u(t+j-1)$ are computed but only the first one (i.e., *u(t)*) is applied to the plant and the same procedure is restarted at *t+1*. This is called the *receding horizon procedure* [Landau, 1998].

The principle of long range predictive control is illustrated in Fig.1.2 where the sequence of desired values y^*, of predicted values $\hat{y}$ and the future control sequence are represented (predicted values are represented for two different future control sequences).

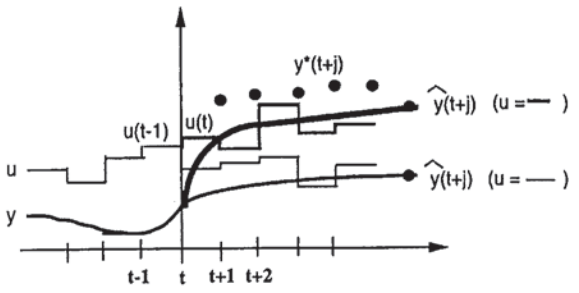


Fig.1.2: Long range predictive control

All the control strategies concern linear design in the sense that on the values of the admissible control applied to the plant are not considered. As a consequence all the control strategies will yield a linear controller of same structure. The use of one or another strategy corresponds finally to different values of the parameters of the controller for the same plant model used for design. Another important issue is that the control should be admissible (realizable) i.e. it should depend only on the information available up to and including time *t* where the control *u(t)* is computed.

The basic System Identification Procedure

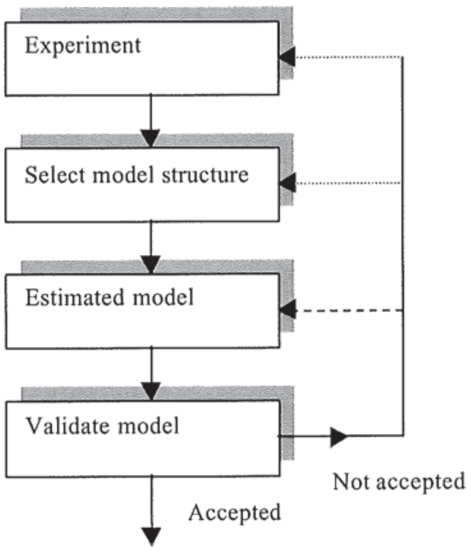


Fig.1.3: The basic system identification procedure.

Some Definitions

Because of the terminologies used in adaptive filtering in automatic control and in the literature on neural networks, are sometimes conflicting therefore the terms that used in this thesis are defined below.

Adaptive vs. non-adaptive training: The training of a network makes use of two sequences, the sequences of inputs and the sequence of corresponding desired outputs. If the network is first trained (with a training sequences of finite length), and subsequently used (with the fixed weights obtained from training), we shall refer to this mode of operation as " non-adaptive." Conversely, the term " adaptive" the mode of operation whereby the network is trained permanently while it is used (with a training sequence of infinite length).

Performance criterion, cost function and training function: The computation of the coefficients during training aims at finding a system whose operation is optimal with respect to some performance criterion which may be either quantitative, e.g., maximising the signal to noise ratio for spatial filtering or qualitative, e.g., the (subjective) quality of speech reconstruction. Assume that we can define a positive *training function,* which is such that a decrease of this function through modifications of the coefficients of the network leads to an improvement of the performance of the system.

In the case of non-adaptive training, the training function is defined as a function of all the data of the training set (in such a case, it is usually termed *cost function);* the minimum of the function corresponds to the optimal performance of the system. Training is an optimization procedure, using gradient-based methods.

In the case of adaptive training, it is impossible, in most instances, to define a time independent cost function whose minimization leads to a system that is optimal with respect to the performance criterion. Therefore, the training function is time-independent. The modification of the coefficients is computed continually from the gradient of the training function. The latter involves the data pertraining to a time window of finite length, which shifts in time (sliding window) and the coefficients are updated at each sampling time for instance.

Recursive vs. non-recursive algorithms, iterative vs. non-iterative algorithms: A *non-recursive* algorithm makes use of a cost function (i.e., a training function defined on a fixed window). A *recursive* algorithm makes use of a training function defined on a sliding window. Therefore, an adaptive system must be trained by a recursive algorithm, where as a non-adaptive system may be trained either by a non-recursive or by a recursive algorithm.

An *iterative algorithm* performs coefficient modifications *several times* from a set of data pertaining to a given time window; a *non-iterative* algorithm does this *only once*. The popular least mean squares (LMS) algorithm is thus a recursive, non-iterative algorithm operating on a sliding window of length 1.