NEAR EAST UNIVERSITY LIBRARY 1988 - LEFKOŞA

NEAR EAST UNIVERSITY

INSTITUTE OF APPLIED AND SOCIAL SCIENCES

DEVELOPMENT OF NEURAL CONTROL SYSTEM

Mehmet Göğebakan

Master Thesis

Department of Computer Engineering

Nicosia-2005

Mehmet Göğebakan : Development of Neural Control System

Approval of the Graduate School of Applied and Social Sciences

Prof. Dr. Fakhraddin Mamedov
Director

We certify this thesis is satisfactory for the award of the Degree of Master of Science in Computer Engineering

Examining Committee in charge:

Prof. Dr. Doğan İbrahim, Chairman , Chairman of Computer Engineering Department, NEU

Assist. Prof. Dr. Kadri Büründcük, Member , Electrical and Electronic Engineering Department, NEU

Assist. Prof. Dr. Firidun Muradov, Member, Computer Engineering Department, NEU

Assoc. Prof. Dr. Rahib Abiyev, Supervisor, Computer Engineering Department, NEU

ACKNOWLEDGEMENTS

Many thanks to my supervisor, Assoc.Prof.Dr Rahib Abiyev for his help and guidance, directing the research program and for his constant encouragement. His challenging questions and imaginative input greatly benefited the work.

I would like to thanks all of my friends who helped me to overcome my project especially Miss Amber Yusuf.

To my fiancé Kadime Altungül I reserve my greatest gratitude for her support and patience during preparation of this thesis.

On a more personal level I must thank: my parents for their support for spurring me on to higher education.

ABSTRACT

Increasing complexity of technological processes, uncertainty of environment leeds to complication the model of control system. For these processes controller developing on the base of traditional approach are very complex and their implementation is difficult. In addition frequently changing environmental conditions of the technological processes force to apply artificial intelligence methodologies with selftraining and adapting ability.

One of these technology that allow to solve above mentioned problem is neural network, that has such property as parallel processing, vitality, self-training capability. These abilities of artificial neural network allow represent non-linear processes, make control system a powerful tool for process modelling and control.

In this thesis the development of neural controller for control of dynamic plant is considered. The structures of industrial neural controllers for technological processes control are represented. The functions of main blocks of neural control system and learning methods are discussed. The main block of neural controllers is neural network. The different models of neurons, which organize neural networks, structure of neural networks and their learning algorithms are described. The development of the control system and its learning algorithms are represented. The simulation of intellectual neural control system for technological process is given.

CONTENTS

LIST OF FIGURES

INTRODUCTION

The model of control system becomes very complicated by increasing complexity of the technological processes, uncertainty of an environment where technological processes take place. For these object the control algorithm developing on base of traditional approach are complex and their implementations are difficult. In addition, the frequently changing of the environmental conditions in the form of unusually disturbance forces to apply artificial intelligence methodologies with self-training and adapting capability. One of these technologies is a neural network. Application of the neural network for constructing control system allows us to increase their computation speed, validity, self-training and adapting capability.

Neural networks, especially in nonlinear system identification and control applications, are typically considered to be blackboxes which are difficult to analyze and understand mathematically. Due to this reason, an indepth mathematical analysis offering insight into the different neural network transformation layers based on a theoretical transformation scheme is desired, but up to now neither available nor known. In previous works it has been shown how proven engineering methods such as dimensional analysis and the Laplace transform may be used to construct a neural network controller topology for time invariant systems. Using the knowledge of neural correspondencies of these two classical methods, the internal nodes of the network could also be successfully interpreted after training.

The objective of this research is to produce an interactive procedure for control system modeling, analysis, synthesis, and design by integrating available classical as well as modern tools such as fuzzy logic, and neural networks. The problem of model free controller design addressed. One approach is to analyze controller design for systems with different nonlinearities and develop a criterion for selecting the appropriate controller. For an unknown system, system classification and identify the system type carried out. Then a controller will be selected based on the analysis. The controller should be capable of self adjustment if the system parameters change.

The application of neural networks has attracted significant attention in several disciplines, such as signal processing, identification and control. The success of neural networks is mainly attributed to their uniqe features:

(1) Parallel structures with distributed storage and processing of massive amounts of information.

(2) Learning ability made possible by adjusting the network interconnection weights and biases based on certain learning algorithms.

The first feature enables neural networks to process large amounts of dimensional information in real-time (e.g. matrix computations), hundreds of times faster than the numerically serial computation performed by a computer. The implication of the second feature is that the nonlinear dynamics of a system can be learned and identified directly by an artificial neural network. The network can also adapt to changes in the environment and make decisions despite uncertainty in operating conditions.

Recent advances in a variety of technologies and applications call for improved performance and reliability, while exacerbating the complexity and uncertainty of systems and their surroundings. In many instances, the operation of systems and devices can be modified and, possibly, optimized by the intervention of a *control system*, that is, an additional mechanism comprised of several components, such as sensors, computers, and actuators that act upon an available input.

The dynamic characteristics and physical properties of the system to be controlled (the plant) can be exploited to design automatic control systems. Some of the main difficulties to be overcome by the designer are the nonlinear plant dynamics and the uncertainties caused by differences between actual and assumed dynamic models. A fixed control design whose performance remains satisfactory in the presence of uncertainty is said to be *robust*. A controller whose parameters vary *on line* during operation is considered to be *adaptive* and can be expected to accommodate for a higher degree of uncertainty than a fixed control structure. If it is capable of adapting to system

failures that are reflected by the state of the plant, then the controller also is *reconfigurable*.

To develop a novel approach for the design of adaptive control systems that are both robust and reconfigurable, and that apply to plants modeled by nonlinear ordinary differential equations. The potential brought about by using postmodern computational paradigms, such as neural networks, in conjunction with conventional control techniques has been recognized. In some cases, a global controller was obtained by training a neural network to approximate the linear gains provided by linear multivariable control.

The aim of the thesis is the development of neural controller for control of technological processes. Thesis consists of introduction, four chapters and conclusion.

In chapter one a review on neural controllers for technological process control is discussed. Introductory remarks on classical, adaptive, non-linear and neural controls are given. The differences of control system based on traditional controllers and neural controller are discussed. A number of research works about neural control systems are briefly described. Problem statement of neural control system is represented.

In chapter two most common types of industrial neural controllers are given. The architecture of neural network based control systems, particularly the structures of the PD, PI, and PID-like neural controllers are presented.

In chapter three, Neural Network structure, its mathematical model and learning of neural control system are represented. The description of backpropagation learning is given. Common transfer functions, supervised and unsupervised learning are explained in details, The Backpropagation algorithm is used for, neural control system learning.

In final chapter the development of neural control system is described. Synthesis of direct neural controller are illustrated, analysis of obtained result are represented. Simulations of neural system have been performed.

Conclusion contains the important results obtained from the thesis.

CHAPTER ONE : A REVIEW ON NEURAL CONTROLLERS FOR TECHNOLOGICAL PROCESS CONTROL

1.1 Overview

This chapter describes a brief introduction about application of neural networks in control systems. The characteristics of classical control are given.

Neural Network application of control systems are described. Breif analysis of each research work is given. The main steps for the development of neural network control system are described.

1.2 Neural correspondencies to classical control theory

Classical Control is based on the use of transfer functions to represent linear differential equations. The classical methods are, however, not readily generalized to multivariable plants, and they do not handle the problem of simultaneously achieving good performance against disturbances as well as robustness against model uncertainties.

Classical control methods enabled the systematic design of early stability augmentation systems, while modern control and robust multi-variable control are critical in all of today's modern flight systems

Basic used classic controllers in industries are P (proportional), PD (proportional derivative), PI (proportional-integral) and PID (proportional-integral) is a simple general-purpose automatic controller that is useful for controlling simple processes.

Proportional –Integral-Derivative (PID) controllers are the most widely used controllers in industries today 90% controllers in industries are PID or PID type of controllers. However, PID has significant limitations:

1. PID works for the process that is basically linear and time-invariant and cannot effectively control complex processes that are nonlinear, time-variant, coupled, and have large time delays, majors disturbances, and uncertainties. Industrial

processes with changing fuels and operating conditions are complex processes for which PID control is insufficient.

2. PID parameters have to be turned properly. If the process dynamics vary due to fuel changes or load changes, PID needs to be re-turned. Tuning PID is often a frustrating and time-consuming experience.
3. PID is a fixed controller, which cannot be used as the the core for a smart control system.

To avoid from these deficiencies, the artificial intelligence ideas are used for constructing controllers, one of them is NN (neural network).

Neural Control is the main interest in neural networks is currently concentrated on the use in adaptive and nonlinear control problems. The need for NNs arises when dealing with non-linear systems for which the linear controllers and models do not satisfy and the use of structures provided by classical control theory seems a straightforward strategy.

Higher demands on the efficiency of processes and technical equipment have introduced a number of applications where classical tuning methods are not sufficient for good performance. At the same time, the availability of inexpensive computing power has made the application of more complex controllers feasible in practice. This development has given rise to a number of new highly challenging application areas for control. In order to give an appreciation of the type of applications involved.

Neural networks' ability to approximate unknown nonlinear mappings with high dimensional input spaces and their potential for on-line learning make them excellent candidates for use in adaptive control systems. Extensive numerical studies have shown that they are capable of dealing with those difficulties typically associated with complex control applications, such as nonlinearity and uncertainty. However, practical applications also call for a better understanding of the theoretical principles involved. In particular, there is no simple way to apply the insights afforded by classical control design methods to the specification and preliminary design of neural network controllers.

Many engineering solutions are tailored to suit linear problems. Generally linear systems pose therefore no unsurmountable problems. In the last years neural networks have mainly been used to model nonlinear systems in control. ANNs (artificial neural networks) can find simple suboptimal solutions to control problems and can be applied to systems where classical approaches based on system linearization do not work. Yet, ANNs lack methods for determining control stability or the possibility to interpret the results analytically. Various approaches and applications of neural control exist in the literature.

Neural control (as well as fuzzy control) was developed in part as a reaction by practitioners to the perceived excessive mathematical intensity and formalism of "classical" control. Although neural control techniques have been inappropriately used in the past, the field of neural control is now starting to mature.

Classical control theory utilizes a number of engineering principles that could be or are partly already applied to neural control. Preliminary studies done have concentrated on neural correspondences of engineering principles used in control and how these principles could be coded into a neural network scheme. It is found that some realizations are clearly straightforward whereas others require more sophisticated procedures which can still be improved.

In the following some of these principles will be explained in brief. Naturally, this list is far from being complete and should only indicate that neural correspondencies to classical engineering principles exist due to the equivalence between a neural topology and a mathematical formulation. These engineering principles are:

• Dimensional analysis.

• The Laplace transform can be used to transform linear differential equations into algebraic equations and though is helpful for the analysis of dynamic systems. It is found that this transformation scheme can be transferred to a neural topology. However, the Laplace transform is only applicable to linear systems.

• The input-output linearization scheme has already been applied to neural control. The basic idea is to identify a feedback which linearizes nonlinear behavior of the system. This way a system can be constructed which can be controlled like a linear system by using the standard classical approaches.

The consequent combination of the above three principles results in a network with predefined layers that do some sort of data pre- and post-processing for a core network that can still be regarded as a black-box and which is the only part to be learned.

Figure 1.1 shows exemplary a schematic diagram of this neural structure. It can be described as a network with a *butterry topology* which indicates that the information processed by the neural network is smaller than the original given data due to an intelligent selection of a data transformation sequence. The modular neural network shown in Figure 1.1 consists of the Pi-transform layers (Π and its inverse Π^{-1}), the Laplace layers (L and its inverse L^{-1}), and the input-output linearization layers (I/O) that encapsulate a core neural network. It should be noted that this structure is not a standard feed-forward network, because the Pi-transform uses shortcuts from the first to the last layer and the linearization requires feedback connections. This structure is found to be suitable for the identification of a dynamic system, which is a basis for many neural control applications.

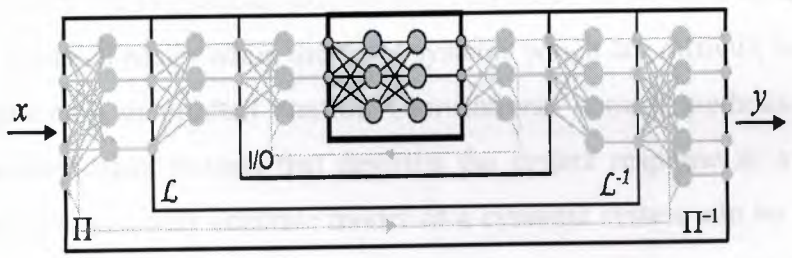


Figure 1.1. Modular neural representation of a nonlinear dynamic system. The buttery diagram: a core network encapsulated by transformation layers. From the inside: core neural network as black-box, input-output linearization layer (I/O), Laplace transform layer, and Pi-transform layer.

Some thoughts should be given to common concepts in control such as the PID controller and the method of gain scheduling. This method provides a linear controller for several linearized states or operational points of a system. It is straightforward to implement gain scheduling in neural networks because the feedback gain coefficient matrices usually are represented as look-up-tables and could as well be stored in neural networks. Even complex designs such as LQR (Linear Quadratic Regulation) or LQG (Linear Quadratic Gaussian) controllers can be implemented most readily into a neural control scheme.

1.3 State of Application Problem of Neural Network Controllers For Technological processes control

Control theory offers powerful tools from linear algebra to be used for system analysis and control as long as the system behaves linearly. Assumptions of system linearity have been made for this reason to develop a control theory on a solid mathematical basis. Control design from system linearization is a widely applied technique in industry. However, in reality most systems are nonlinear. It is the ability of neural networks to model nonlinear systems which is the feature most readily exploited in the synthesis of nonlinear controllers.

Dynamic systems are in general complex and nonlinear. Some systems are simplified in order to be modeled easily while there are systems which are difficult to model such as a process planning or product control. Conventional control methods are in general based on mathematical models that describe the system response as a function of its inputs. Even if a relatively accurate model of a dynamic system can be developed, it is often too complex to be used in controller development. Thus, model free controllers, specifically PIDs are widely used in practice. The usual approach to optimize the system performance controlled by a PID is to tune the PID coefficients. This approach may be acceptable as long as the system parameters are not varying or do not display nonlinearities. Some robust control methods, such as H-infinity, have been developed to deal with parametric uncertainties and disturbances. But they still require a low order of the system and knowledge of the disturbance variations, and they are computationally

demanding. An alternative approach to control complex, nonlinear, and ill-defined systems is the use of modern tools such as fuzzy logic, and neural networks

The first application of neural networks to control systems was developed in the mid-1980s. Models of dynamic systems and their inverses have immediate utility in control. In the literature on neural networks, architectures for the control and indentification of a large number of control structures have been proposed and used [1]. Some of the well-established and well-analyzed structures which have been applied in guidance and control designs are described below. Note that some network schemes have not been applied in this field but do possess potential are also introduced in the follows.

Neural control techniques have successfully been applied to problems in robotics and other highly nonlinear systems. A growing number of different neural control schemes exist that are fitted only for certain problems. However, the usage of neural networks in nonlinear control does not make sense *per se*. There are still many open research topics, such as the characterization of theoretical properties such as stability, controllability and observability or even the system identifiability.

It is not intended to give a survey on neural control methods here, since many of the basic principles are shown in the reports by Hunt [2] or Narendra [1]. The idea of a neural network structural compiler originates from the (re-)use of existing control theory applications, which intends the construction of mathematical controllers designed after classical theories and their representation in the form of neural networks.

Therefore it is claimed that it will be possible to design neural controllers at least as good as the classical ones. However, by providing the network with additional degrees of freedom and applying training algorithms common in neural network computation, even an improved adaption could be achieved. Adaptivity is an important feature because the real world environment of the controller can be expected to be different from the simplified linearized model used for the controller design.

Analogous approaches to the idea promoted here already exist in techniques summarized under the term of intelligent control which represents an attempt to bring

together artificial intelligence techniques and control theory. Controllers are put together from predefines components in a structured design approach with a knowledge-based expert system as integration tool.

This is realized for instance in the neuro-fuzzy control scheme. A structure is provided by the fuzzy logic approach which builds up control laws from linguistic rules. Then the scheme is implemented in a neural network. The structure is determined after a simple algorithm from modules. Finally, the learning ability of the neural network is used to adapt the controller to the specific control situation by learning the controller's parameter values.

There are number of research and work about development control system on the base of neural networks:

In [3], classical and neural control systems are synthesized to combine the most effective elements of old and new design concepts with the promise of producing better control systems. The novel approach to nonlinear control design retains the characteristics of stability and robustness of classical, linear control laws, while capitalizing on the broader capabilities of a so-called *adaptive critic* neural network. First, the neural control system's architecture and parameters are determined from the initial specification of the control law by solving algebraic linear systems of equations during a so-called *pretraining phase*. Secondly, the neural parameters are modified during an *on-line training phase* to account for uncertainties that were not captured in the linearizations, such as nonlinear effects, control failures, and parameter variations.

In [4], the development of neural controllers for control of dynamic plants by using recurrent neural network is considered. The structure and learning algorithm of the recurrent neural network are described. Using learning algorithm and desired time response characteristics of the system the synthesis of neural controller for technological process control is carried out.

Also using fuzzy models of the control object and desired time response characteristic of system the synthesis of fuzzy neural controller is carried out. The learning of fuzzy

neural controller is performed by using α – level procedure and interval arithmetic. The simulation of fuzzy control system is performed and result of simulation are described.

In [5] provided a quick overview of neural networks and to explain how they can be used in control systems. Then introduced the multilayer perceptron neural network and describe how it can be used for function approximation.

The backpropagation algorithm (including its variations) is the principal procedure for training multilayer perceptrons; it is briefly described here. Care must be taken, when training perceptron networks, to ensure that they do not overfit the training data and then fail to generalize well in new situations. Several techniques for improving generalization are discussed. Three control architectures are presented: model reference adaptive control, model predictive control, and feedback linearization control. These controllers demonstrate the variety of ways in which multilayer perceptron neural networks can be used as basic building blocks. Next demonstrated the practical implementation of these controllers on three applications: a continuous stirred tank reactor, a robot arm, and a magnetic levitation system.

Dynamical control problems and the application of artificial neural networks to solve optimization are discussed in [6], A general framework for artificial neural networks models is introduced first. Then the main feedforward and feedback models are presented. The IAC (Interactive Activation and Competition) feedback network is analyzed in detail. It is shown that the IAC network, like the Hopfield network, can be used to solve quadratic optimization problems.

A method that speeds up the training of feedforward artificial neural networks by constraining the location of the decision surfaces defined by the weights arriving at the hidden units is developed [6].

The problem of training artificial neural networks to be fault tolerant to loss of hidden units is mathematically analyzed. It is shown that by considering the network fault tolerance the above problem is regularized, that is the number of local minima is reduced. It is also shown that in some cases there is a unique set of weights that

minimizes a cost function. The BPS algorithm, a network training algorithm that switches the hidden units on and off, is developed and it is shown that its use results in fault tolerant neural networks.

A novel non-standard artificial neural network model is then proposed to solve the extremum control problem for static systems that have an asymmetric performance index. An algorithm to train such a network is developed and it is shown that the proposed network structure can also be applied to the multi-input case.

A control structure that integrates feedback control and a feedforward artificial neural network to perform nonlinear control is proposed. It is shown that such a structure performs closed-loop identification of the inverse dynamical system. The technique of adapting the gains of the feedback controller during training is then introduced. Finally it is shown that the BPS algorithm can also be used in this case to increase the fault tolerance of the neural controller in relation to loss of hidden units. Computer simulations are used throughout to illustrate the results.

In [7] addressed two neural network based control systems. The first is a neural network based predictive controller. System identification and controller design are discussed. The second is a direct neural network controller. Parameter choice and training methods are discussed. Both controllers are tested on two divergent plants. Problems regarding implementations are discussed. First the neural network based predictive controller is introduced as an extension to the generalised predictive controller (GPC) to allow control of nonlinear plant. The controller design includes the GPC parameters, but prediction is done explicitly by using a neural network model of the plant.

System identification is discussed. Two control systems are constructed for two divergent plants: A coupled tank system and an inverse pendulum. This shows how implementation aspects such as plant excitation during system identification are handled. Limitations of the controller type are discussed and shown on the two implementations.

In the second part of [7], the direct neural network controller is discussed. An output feedback controller is constructed around a neural network. Controller parameters are determined using system simulations.

The control system is applied as a single step ahead controller to two different plants. One of them is a path following problem in connection with a reversing trailer truck. This system illustrates an approach with stepwise increasing controller complexity to handle the unstable control object. The second plant is a coupled tank system. Comparison is made with the first controller. Both controllers are shown to work. But for the neural network based predictive controller, construction of a neural network model of high accuracy is critical – especially when long prediction horizons are needed. This limits application to plants that can be modelled to sufficient accuracy.

The direct neural network controller does not need a model. Instead the controller is trained on simulation runs of the plant. This requires careful selection of training scenarios, as these scenarios have impact on the performance of the controller.

As a further extension to these works, [8] describes the latest results of a theoretical interpretation framework describing the neural network transformation sequences in nonlinear system identification and control. This can be achieved by incorporation of the method of exact input-output linearization in the above mentioned two transformation sequences of dimensional analysis and the Laplace transformation. Based on these three theoretical considerations neural network topologies may be designed in special situations by a pure *translation* in the sense of a structural compilation of the known classical solutions into their correspondent neural topology.

Based on known exemplary results, in [8] a *structural compiler for neural networks* is proposed. This structural compiler for neural networks is intended to automatically convert classical control formulations into their equivalent neural network structure based on the principles of equivalence between formula and operator, and operator and structure which are discussed in detail in this work.

In [9] proposed a learning scheme for a neuro control system with a Control Network and an Identification Network. For the Identification Network, the learning scheme is the popular backpropagation. For the Control Network, the Plant Information is calculated on-line and fed along with other inputs to train the network On-line

simulation studies and experimental results for selected process with the proposed control are presented and discussed.

A systematic approach is developed for designing adaptive and reconfigurable nonlinear control systems that are applicable to plants modeled by ordinary differential equations. The nonlinear controller comprising a network of neural networks is taught using a two-phase learning procedure realized through novel techniques for initialization, on-line training, and adaptive critic design. A critical observation is that the gradients of the functions defined by the neural networks must equal corresponding linear gain matrices at chosen operating points. On-line training is based on a dual heuristic adaptive critic architecture that improves control for large, coupled motions by accounting for actual plant dynamics and nonlinear effects. An action network computes the optimal control law; a critic network predicts the derivative of the cost-to-go with respect to the state. Both networks are algebraically initialized based on prior knowledge of satisfactory pointwise linear controllers and continue to adapt on line during full-scale simulations of the plant.

On-line training takes place sequentially over discrete periods of time and involves several numerical procedures. A backpropagating algorithm called Resilient Backpropagation is modified and successfully implemented to meet these objectives, without excessive computational expense. This adaptive controller is as conservative as the linear designs and as effective as a global nonlinear controller. The method is successfully implemented for the full-envelope control of a six-degree-of-freedom aircraft simulation. The results show that the on-line adaptation brings about improved performance with respect to the initialization phase during aircraft maneuvers that involve large-angle and coupled dynamics, and parameter variations.

In [10] some novel applications of neural networks in process control presented. Four different approaches utilizing neural networks are presented as case studies of nonlinear chemical processes. It is concluded that the hybrid methods utilizing neural networks are very promising for the control of nonlinear and/or Multi-Input Multi-Output systems which can not be controlled successfully by conventional techniques.

In [11], linear and non-linear analysis of rectangular plates has been presented via artificial intelligence techniques and numerical examples are solved by means of the developed program. The back-propagation neural network has been used in the solution. The thickness of plates has been normalized by the use of the fuzzy triangular membership function. The center point moments and deflection have been obtained for the numerical applications. It has been emphasized that the artificial intelligence technique is an alternative method that can be used in structural engineering.

In [12], the development of Intellectual Systems for Technological Processes Control is considered. The application of Artificial Neural Systems for solving control problems is given. The main blocks of Neural Control Systems are analyzed; their structure and learning methods are discussed. The different models of neurons, which organize Neural Networks, structure of Neural Networks and their learning algorithms, are described. The development of the control system on the base of Recurrent Neural Networks is shown and its learning algorithms are widely described on the base of "Back Propagation", such as Back Propagation for fully Recurrent Neural Network, Back Propagation for Multilayered Recurrent Neural Networks or Back Propagation in time. Using described learning algorithms, the structure of intellectual Neural Control Systems for Technological Process is given. The synthesis and modeling of this system are described.

A mobile robot whose behavior is controlled by a structured hierarchical email network and its teaming algorithm is presented [13]. The robot has four wheels and moves about freely with two motors. Twelve or more sensors are used to monitor internal conditions and environmental changes. These sentimental are presented to the input layer of the network, and the output is used as motor control signals. The network model is divided into two sub-networks connected to each other by short-term memory units used to process time-dependent data. A robot can be taught behaviors by changing the patterns presented to it. For example, a group of robots were taught to play a cops-and-robbers web. Through training, the robots learned them such as capture and escape. Similarly, other types of robots are used to work as a housewife, like for example working in the kitchen, washing dishes, cooking food, etc. These robots learn by looking at the

examples and feeding them in their memory. Thus, by this way their characteristics are similar to the human beings.

Neural networks can be used effectively for the identification and control of nonlinear dynamical systems. The emphasis of the part is on models for both identification and control. Static and dynamic back-propagation methods for the adjustment of parameters are discussed. In the models that are introduced, multilayer and recurrent networks are interconnected in novel configurations and hence there is a real need to study them in an unfilled fashion. Simulation results reveal that the identification and adaptive control schemes suggested are practically feasible. Basic concepts and definitions are introduced and theoretical questions, which have to be addressed, are also described [14].

In a medical ultrasound imaging system the control parameters for the beam former are usually designed based on a constant sound velocity for the tissue. The velocity in the intervening tissues (the body wall) can vary by as much as 8%, leading to a spurious echo delay noise across the array. This has a detrimental effect on the image quality. Since the delay noise is not deterministic, its effects can not be pre-compensated in the beam former subsystem. Degradation of image quality caused by delay noise can be quantified in terms of the changes in the imaging point-spread-function (PSF). A major engineering challenge in medical ultrasound, which remains, is the conception of a real time, adaptive technique for delay noise removal to improve the image quality. Flax and O'Donnell have reported a method based on the cross correlation of A-lines for adaptive image restoration. Nock Efal, have described a method which utilizes the speckle brightness as a quality factor feedback for adaptive changing of the relative delays between channels. Fink of al., has recently described a time reversal method based on ideas from adaptive optics [15].

Artificial neural network (ANN) approaches to electric load forecasting is given in [16]. The ANN is used to learn the relationship among past, current and future temperatures and loads. In order to provide the forecasted load, the ANN interpolates among the load and temperature data in a training data set. The average absolute errors of the one-hour and 24-hour ahead forecasts in our test on actual utility data are shown to be 1.40% and 2.06%, respectively. This compares with an average error of 4.22% for 24-hour ahead

forecasts with a currently used forecasting technique applied to the same data. Various techniques for power system load forecasting have been proposed in the last few decades. Load forecasting with lead-times, from a few minutes to several days, helps the system operator to efficiently schedule spinning serve allocation. In addition, load forecasting can provide information, which can be used, for possible energy interchange with other utilities. In addition to these economical reasons, load forecasting is also useful for system security. If applied to the system security assessment problem, it can provide valuable information to detect many vulnerable situations in advance [16].

An Artificial Neural Network has been Implemented In the Explosives Detection Systems fielded at various airports [17]. Tests of the on-line performance of the Neural Network (NN) confirmed its superiority over standard statistical techniques, and the NN was installed as the decision algorithm. Analysis of the mass of data being produced is still underway; but preliminary conclusions are presented [17].

The Neural Network technique was applied to the same features used by the discriminant analysis. These features were combinations of the signals from the detector array, such as the total nitrogen content of the bag, maximum intensity in the reconstructed three dimensional image, et al. These features have different statistical properties, and different amounts of information about the presence or absence of explosives in the bag. Combinations of these features provide the discriminant value, which is used to decide whether or not there is a threat in the bag. Because of the success of the standard analysis, the problem is known to be solvable; and, in fact, there is a target to be beat.

Automatic speech recognizers currently perform poorly in the presence of noise. Humans, on the other hand, often compensate for noise degradation by extracting speech information from alternative sources and then integrating this information with the acoustical signal. Visual signals from the speaker's face are one source of supplemental speech information. It is demonstrated that multiple sources of speech information can be integrated at a sub symbolic level to improve vowel recognition.

Feedforward and recurrent neural networks are trained to estimate the acoustic characteristics; of the vocal tract from images of the speaker's mouth. These estimates

are then combined with the noise-degraded acoustic information, effectively increasing the signal-to-noise ratio and improving the recognition of these noise-degraded signals. Alternative symbolic strategies, such as direct categorization of the visual signals into vowels, are also presented. The performances of these neural networks compared favorably with human performance and with other pattern-matching and estimation techniques [18].

Communication by using the acoustic speech signal alone is possible, but often communication also involves visible gestures from the speaker's face and body. in situations where environmental noise is present or the listener is hearing impaired, these visual sources of information become crucial to understanding what has been said. Our ability to comprehend speech with relative ease under a wide range of environmental circumstances is due largely to our ability to fuse multiple sources of information in real time.

Loss of information in the acoustic signal can be compensated by using information about speech articulation from the movements around the mouth. This work was supported by using semantic information conveyed by facial expressions and other gestures. At the same time, the listener can use knowledge of linguistic constraints to further compensate for ambiguities remaining in the received speech signals [18].

1.4 Problem Statement

The application of NN for controlling of technological processes allows to increase the quality of the control. To develop neural control system the following have been carried out within this thesis:

1. To Develop Structure of Neural Control System.
2. Describe Learning of Neural Control System
3. Develop Neural Control System
4. Computer (simulation) modeling Neural Control System

1.5 Summary

There are many different approaches of using neural networks in control systems.

Result of analysis different research works about neural control system shows that they have good performance for adaptive controlling. Neural network control system is applied to different non-linear dynamic processes to improve accuracy of the control. For this reason the development of neural network control system for non-linear process began actual. The applications areas of NN and state of art of neural control system are given.

CHAPTER TWO: ARCHITECTURE OF NEURAL NETWORK BASED CONTROL SYSTEM

2.1 Overview

Application of neural network for constructing control system allows to increase their computation speed, validity, self-training and adapting capability.

In this chapter the development of controllers based on Neural Networks are considered. The structures of PD, PI and PID like neural controllers are given. The functions of their main block have been explained.

2.2 Components of Industrial controller

The more used controllers in industries are PD, PI and PID like controller. PID stands for Proportional, Integral, Derivative. Controllers are designed to eliminate the need for continuous operator attention. Cruise control in a car and a house thermostat are common examples of how controllers are used automatically adjust some variable to hold the measurement (or process variable) at the set-point. Error is defined as the difference between set-point and measurement. (error) = (set-point) - (measurement). The variable being adjusted is called the manipulated variable which usually is equal to the output of the controller. The output of PID controllers will change in response to a change in measurement or set-point. Manufacturers of PID controllers use different names to identify the three modes. These equations show the relationships:

```
P    Proportional Band = 100/gain
I              Integral = 1/reset          (units of time)
D           Derivative = rate = pre-act    (units of time)
```

Depending on the manufacturer, integral or reset action is set in either time/repeat or repeat/time. One is just the reciprocal of the other. Note that manufacturers are not consistent and often use reset in units of time/repeat or integral in units of repeats/time. Derivative and rate are the same.

With proportional band, the controller output is proportional to the error or a change in measurement (depending on the controller).

(controller output) = (error)*100/(proportional band)

With a proportional controller offset (deviation from set-point) is present. Increasing the controller gain will make the loop go unstable. Integral action was included in controllers to eliminate this offset.

With integral action, the controller output is proportional to the amount of time the error is present. Integral action eliminates offset.

CONTROLLER OUTPUT = (1/INTEGRAL) (Integral of) e(t) d(t)

Notice that the offset (deviation from set-point) in the time response plots is now gone. Integral action has eliminated the offset. The response is somewhat oscillatory and can be stabilized some by adding derivative action.

Integral action gives the controller a large gain at low frequencies that results in eliminating offset and "beating down" load disturbances. The controller phase starts out at –90 degrees and increases to near 0 degrees at the break frequency. This additional phase lag is what we give up by adding integral action. Derivative action adds phase lead and is used to compensate for the lag introduced by integral action.

With derivative action,the controller output is proportional to the rate of change of the measurement or error. The controller output is calculated by the rate of change of the measurement with time.

$$\text{CONTROLLER OUTPUT} = \text{DERIVATIVE} \frac{dm}{dt}$$

Where m is the measurement at time t.

Some manufacturers use the term rate or pre-act instead of derivative. Derivative, rate, and pre-act are the same thing.

DERIVATIVE = RATE = PRE ACT

Derivative action can compensate for a changing measurement. Thus derivative takes action to inhibit more rapid changes of the measurement than proportional action. When a load or set-point change occurs, the derivative action causes the controller gain to move the "wrong" way when the measurement gets near the set-point. Derivative is often used to avoid overshoot.

Derivative action can stabilize loops since it adds phase lead. Generally, if you use derivative action, more controller gain and reset can be used.

With a PID controller the amplitude ratio now has a dip near the center of the frequency response. Integral action gives the controller high gain at low frequencies, and derivative action causes the gain to start rising after the "dip". At higher frequencies the filter on derivative action limits the derivative action. At very high frequencies (above 314 radians/time; the Nyquist frequency) the controller phase and amplitude ratio increase and decrease quite a bit because of discrete sampling. If the controller had no filter the controller amplitude ratio would steadily increase at high frequencies up to the Nyquist frequency (1/2 the sampling frequency). The controller phase now has a hump due to the derivative lead action and filtering.

The time response is less oscillatory than with the PI controller. Derivative action has helped stabilize the loop.

It is important to keep in mind that understanding the process is fundamental to getting a well designed control loop. Sensors must be in appropriate locations and valves must be sized correctly with appropriate trim.

In general, for the tightest loop control, the dynamic controller gain should be as high as possible without causing the loop to be unstable.

P is in units of proportional band. I is in units of time/repeat. So increasing P or I, *decreases* their action in the picture.

2.3 Structure of the PD-Like Controller

Even though at present traditional P, PD, PI, PID controls are still widely used in industrial control systems, their ability to cope with some complex process properties such as non-linearities, time-varying parameters and long time delays are known to be very poor. Systems based on neural network one of tolls that could deal with these problems.

In Figure 2.1 the structure of neural PD control system is shown. The output signal of the control object y(t) is compared with the target signal G(t) of the system and the value of the error signal e(t) is passed to the differentiator D. The output signal of differentiator e'(t) and error signal e(t) after multiplying to the scaling coefficients ke and ke' are entered to the neural network input. These coming input signals after processing the output of neural network scaled and output control action is tranfered to the input of control object.

The synthesis of neural network controller includes the determination of the scale coefficients and parameters of the neural network (NN). In the controller synthesis processes the main problem is learning of the NN coefficients. Assume there is target behaviour for the constructed control system. It is necessary to determine the values of parameters- weights matrix w_{ij} and scale coefficient k_e, $k_{e'}$, k_u using of which in control system for object (2.1) would allow to achive time response which provides target step response of the system

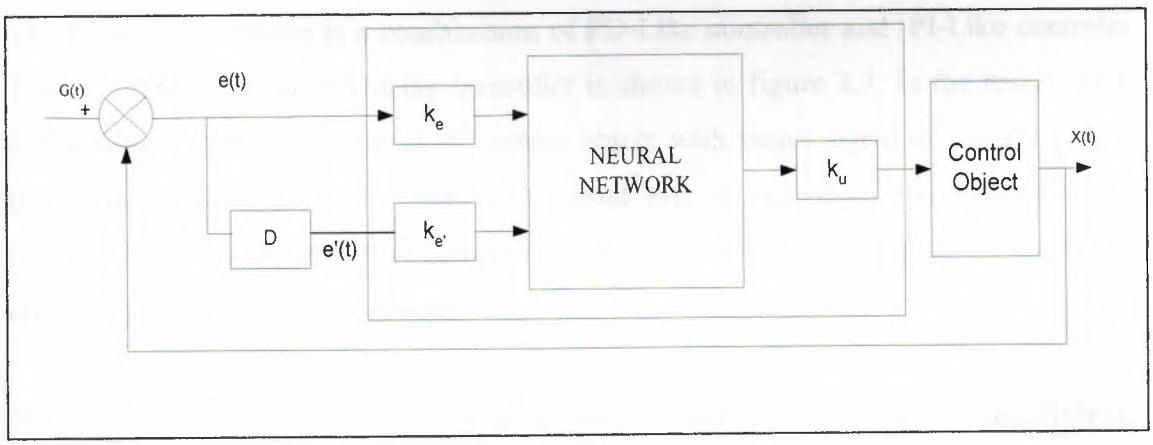


Figure 2.1 Structure of PD control system

2.4 Structure of the PI-Like Controller

The structure of the neural PI-like controller is shown in Figure 2.2. The output signal of control object is compared with the target signal G(t) in the comparator. In the result of comparision the value of error between target and current signals of control object is determined. This signal e(t) is passed to integrator $\int$ and the integral value of error is determined. The error signal e(t) and the integral value of error $\int$ e(t) after multiplying to the scaling coefficients k_e and $k_\int$ are entered to the scaling block, where after scaling they are entered to the input of neural network.

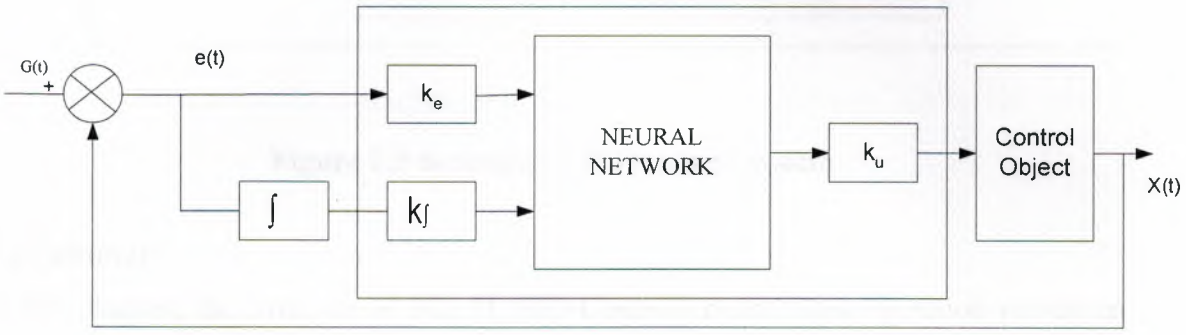


Figure 2.2 Structure of PI control system

Using neural network block the output of the controller is determined. This output signal after scaling is entered to the plant input

2.5 Structure of The PID-Like Controller

The PID-Like controler is a combination of PD-Like controller and PI-Like controler. The structure of neural PID-Like controller is shown in figure 2.3. In the result of the comparison of the output signal of control object with target signal of control system the value of error is determined. The signal e(t) is passed to the integrator and differentiator. On the output of integrator and differentiator the integral value of error and change of error are determined.

The error signal e(t), velocity e`(t) and the integral value of error $\int$e(t) after multiplying to the scaling coefficients ke, ke' and $k_\int$ are entered to the neural network block. Using

neural network block the output of the controller is determined. This output signal after scaling is entered to the plant input.

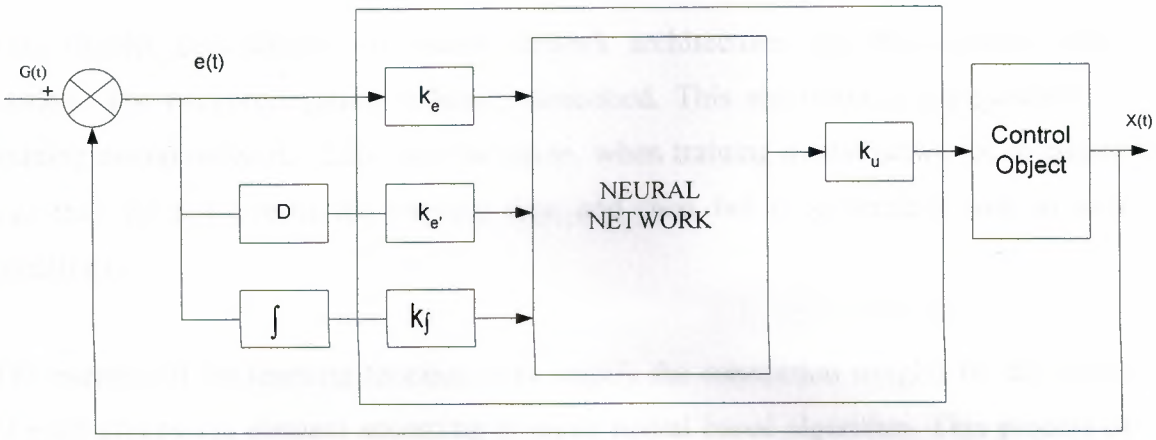


Figure 2.3 Structure of PID control system

2.6 Summary

In this chapter, the structure of PD, PI, PID Controllers and their operation principles are given. The structures of PI, PD, PID Like Neuro Controllers and their functions are explained.

CHAPTER THREE: LEARNING OF NEURAL CONTROL SYSTEM

3.1 Overview

This chapter concentrates on neural network architectures and their mathematical models. The backpropagation is briefly described. This algorithm is a procedure for training neural network. Care must be taken, when training neural networks, to ensure that they do not overfit the training data and then fail to generalize well in new situations.

The purpose of the learning function is to modify the connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the adaption function, as well as the learning mode.

There are two types of learning: supervised and unsupervised. They are discussed in detail in this chapter.

3.2 Neural Network Structure and Its Mathematical Model

Neural Network consists of set of neurons. In Figure 3.1 the mathematical model of neuron that have multi-input is shown.

The scalar inputs are multiplied by the scalar *weight* to form, one of the terms that is sent to the summer. The other input, 1, is multiplied by a *bias* and then passed to the summer. The summer output, often referred to as the *net input*, goes into a *transfer function,* which produces the scalar neuron output.

Typically, a neuron has more than one input. A neuron with inputs R is shown in Figure 3.1 The individual inputs $p_1 p_2, \ldots, p_R$ are each weighted by corresponding elements of $w_{1,1}.w_{1,2} w_{1,R}$ the *weight matrix* **W**

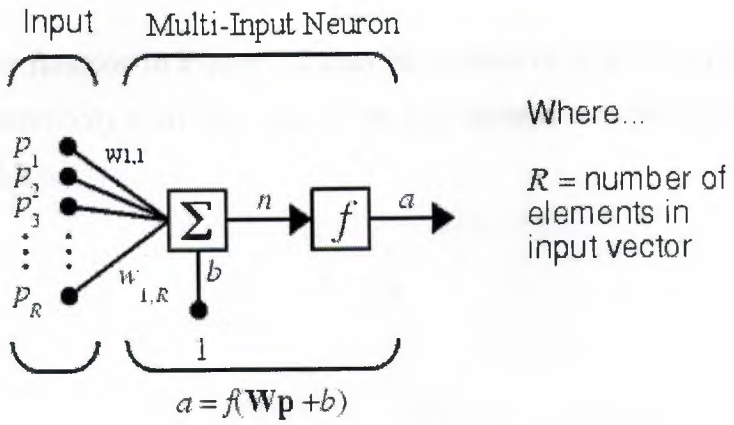


Figure 3.1 Multi-Input Neuron

The neuron has a bias b, which is summed with the weighted inputs to form the net input : *n*

$$n = w_{1,1} p_1 + w_{1,2} p_2 + \ldots + w_{1,R} p_R + b.$$

This expression can be written in matrix form:

$$n = Wp + b,$$

where the matrix **W** for the single neuron case has only one row. Now the neuron output can be written as

$$a = f(Wp + b) .$$

Figure 3.2 represents the neuron in matrix form.

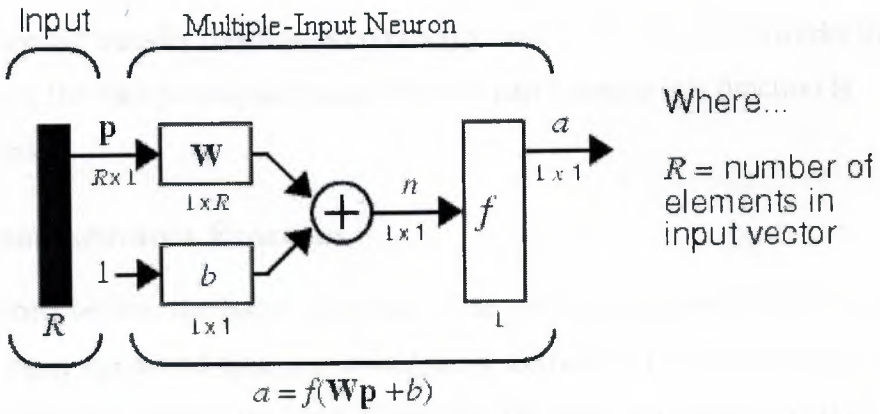


Figure 3.2 Neuron *R* with Inputs, Matrix Notation

Note that w and b are both *adjustable* scalar parameters of the neuron. Typically the transfer functions are chosen by the designer, and then the parameters are adjusted by some learning rule so that the neuron input/output relationship meets some specific goal.

The transfer function in Figure 3.2 may be a linear or a nonlinear function of n. One of the most commonly used functions is the *log-sigmoid transfer function*, which is shown in Figure 3.3

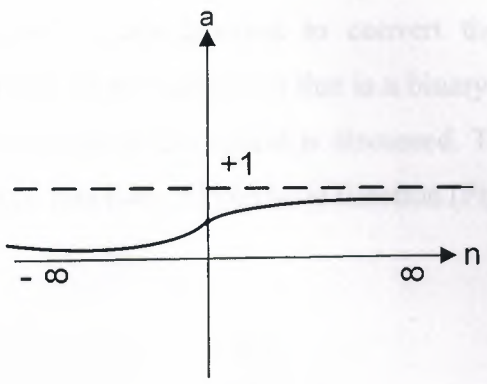


Figure 3.3 Log-Sigmoid Transfer Function

This transfer function takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 to 1, according to the expression:

$$a = \frac{1}{1+e^{-n}}$$

The log-sigmoid transfer function is commonly used in multilayer networks that are trained using the backpropagation algorithm, in part because this function is differentiable.

3.3 Common Activation Functions

As mentioned before, the basic operation of an artificial neuron involves summing its weighted input signal and applying an output, or activation function. For the input units, this function is the identity function. Typically, the same activation function is used for all neurons in any particular layer of a neural net, although this is not required. In most cases, a nonlinear activation function is used. In order to achieve the advantages of multilayer nets, compared with the limited capabilities of single-layer nets, nonlinear functions are required (since the results of feeding a signal through two or more layers

of linear processing elements-i.e. , elements with linear activation functions are no different from what can be obtained using a single layer).

(i) Identity function:

$$f(x) = x \qquad \text{for all } x .$$

Single-layer nets often use a step function to convert the net input, which is a continuously valued variable, to an output unit that is a binary (1 or 0) or bipolar (1 or -1) signal. The use of a *threshold* in this regard is discussed. The binary step function is also known as the threshold function or Heaviside function.(Figure 3.4)

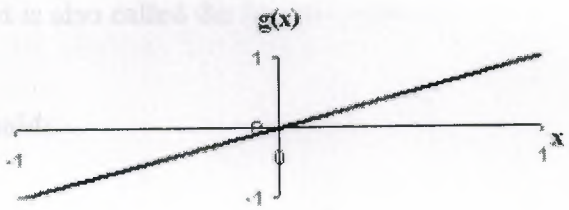


Figure 3.4 Identity function.

(ii) Binary step function (with threshold θ):

$$f(x) = \begin{cases} 1 \ldots\ldots\ldots\ldots if \ldots x \geq \theta \\ 0 \ldots\ldots\ldots\ldots if \ldots x < \theta \end{cases}$$

Sigmoid functions (S-shaped curves) are useful activation functions. The logistic function and the hyperbolic tangent functions are the most common. They are especially advantageous for use in neural nets trained by backpropagation, because the simple relationship between the value of the function at a point and the value of the derivative at that point reduces the computational burden during training.(Figure 3.5)

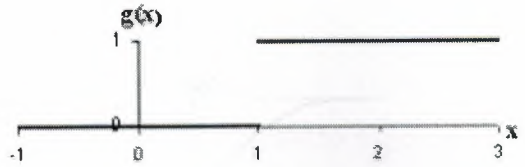


Figure 3.5 Binary step function

The logistic function, a sigmoid function with range from 0 to 1, is often used as the activation function for neural nets in which the desired output values either are binary or are in the interval between 0 and 1. To emphasize the range of the function, we will call it the *binary sigmoid;* it is also called the *logistic sigmoid*

(iii) Binary sigmoid:

$$f(x) = \frac{1}{1 + \exp(\sigma x)}$$

$$f'(x) = \sigma f(x)[1 - f(x)]$$

As the logistic sigmoid function can be scaled to have any range of values that is appropriate for a given problem. The most common range is from - 1 to 1, we call this sigmoid the *bipolar sigmoid.* It is illustrated in Figure 3.6 for $\sigma = 1$. ?

(iv) Bipolar sigmoid:

$$g(x) = 2f(x) - 1 = \frac{2}{1 + \exp(-\sigma x)} - 1$$

$$= \frac{1 - \exp(-\sigma x)}{1 + \exp(-\sigma x)}$$

$$g'(x) = \frac{\sigma}{2}[1 + g(x)][1 - g(x)].$$

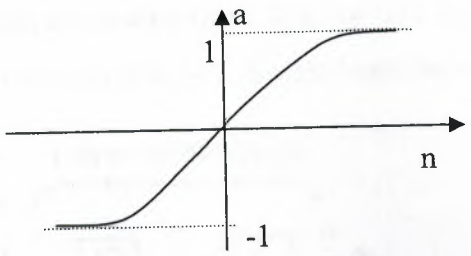


Figure 3.6. Bipolar sigmoid

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is also often used as the activation function when the desired range of output values is between - 1 and 1. We illustrate the correspondence between the two for σ=1 . We have

$$g(x) = \frac{1-\exp(-x)}{1+\exp(-x)}$$

The hyperbolic tangent is

$$h(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

$$= \frac{1-\exp(-2x)}{1+\exp(-2x)}$$

The derivative of the hyperbolic tangent is:

$$h'(x) = [I + h(x)][I - h(x)].$$

For binary data (rather than continuously valued data in the range from 0 to 1), it is usually preferable to convert to bipolar form and use the bipolar sigmoid or hyperbolic tangent.

3.4 Network Architectures

Commonly one neuron, even with many inputs, is not sufficient. We might need five or ten, operating in parallel, in what is called a *layer*. A single-layer network of neurons is shown in Figure 3.7. Note that each of the inputs is connected to each of the neurons

and that the weight matrix now has rows. The layer includes the weight matrix, the summers, the bias vector, the transfer function boxes and the output vector. Some authors refer to the inputs as another layer, but we will not do that here. It is common for the number of inputs to a layer to be different from the number of neurons

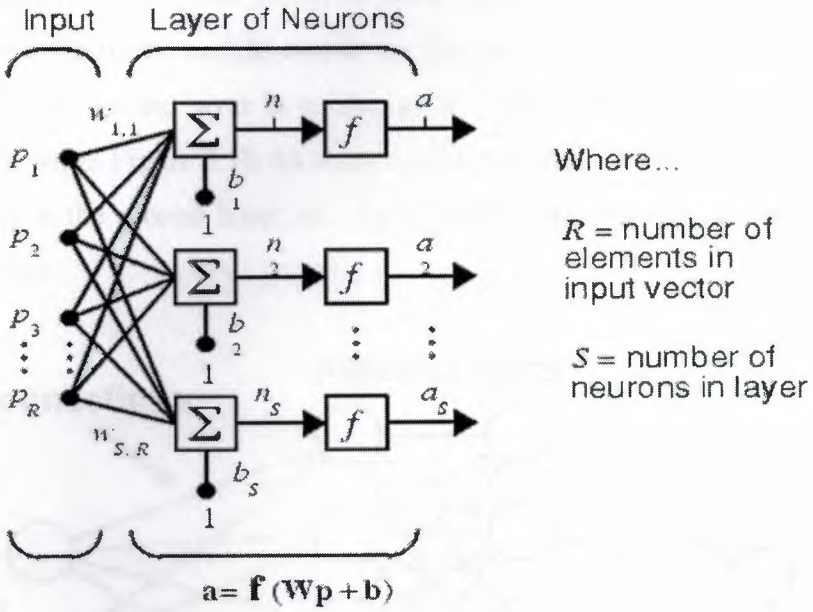


Figure 3.7 Layer of *S* Neurons

The *S*-neuron, *R*-input, one-layer network also can be drawn in matrix notation, as shown in Figure 3.8.

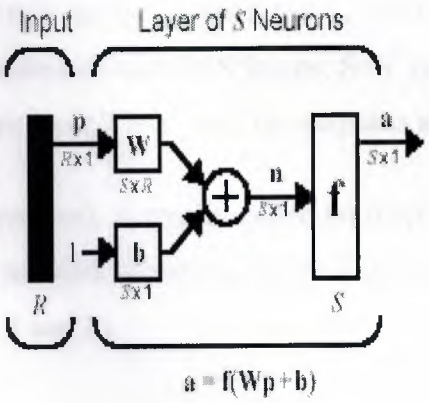


Figure 3.8 Layer *S* of Neurons, Matrix Notation

3.4.1 Multiple Layers of Neurons

Now consider a network with several layers. Each layer has its own weight matrix , its own bias vector, a net input vector and an output vector . We need to introduce some additional notation to distinguish between these layers. We will use superscripts to identify the layers. Thus, the weight matrix for the first layer is written as W^1, and the weight matrix for the second layer is written as W^2. This notation is used in the three-layer network shown in Figure 3.10 As shown, there are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer, etc. As noted, different layers can have different numbers of neurons.

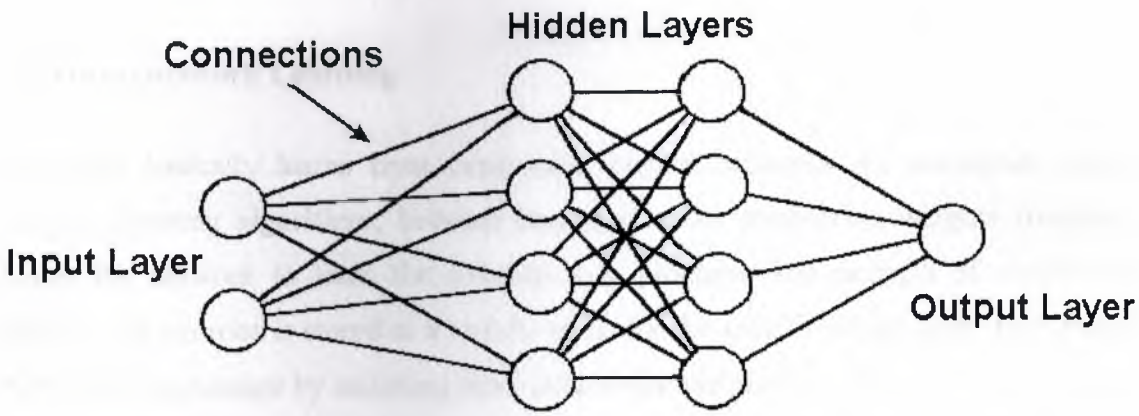


Figure 3.9 Multilayer Neural Network

The outputs of layers one and two are the inputs for layers two and three. Thus layer 2 can be viewed as a one-layer network with $R=S^1$inputs, $S=S^2$ neurons, and an S^2xS^1 weight matrix W^2. The input to layer 2 is a^1, and the output is a^2

A layer whose output is the network output is called an *output layer*. The other layers are called *hidden layers*. The network shown in Figure 3.10 has an output layer (layer 3) and two hidden layers (layers 1 and 2).

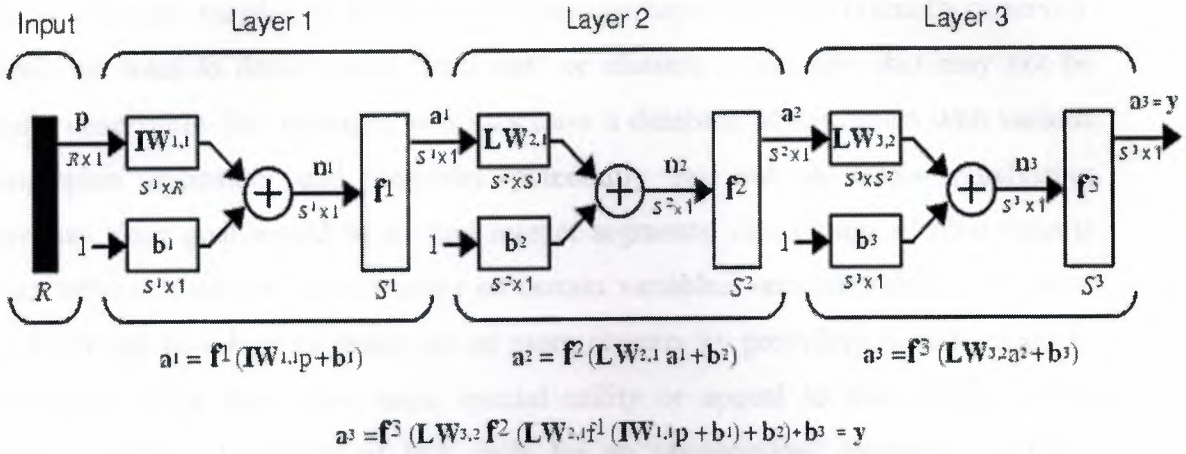


Figure 3.10 Three-Layer Network

3.5 Neural network Learning

The brain basically learns from experience. Neural networks are sometimes called machine-learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights.

The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training.

The training method usually consists of one of three schemes:

3.5.1 Unsupervised Learning in Neural Networks

Training algorithms that adjust the weights in a neural network by reference to a training data set including input variables only. Unsupervised learning algorithms attempt to locate clusters in the input data.

The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing.

Here the outcome variable of interest is not (and perhaps cannot be) directly observed. Instead, we want to detect some "structure" or clusters in the data that may not be trivially observable. For example, you may have a database of customers with various demographic indicators and variables potentially relevant to future purchasing behaviour. Your goal would be to find market segments, i.e., groups of observations that are relatively similar to each other on certain variables; once identified, you could then determine how best to reach one or more clusters by providing certain goods or services you think may have some special utility or appeal to individuals in that segment (cluster). This type of task calls for an unsupervised learning algorithm, because learning (fitting of models) in this case cannot be guided by previously known classifications. Only after identifying certain clusters you can begin to assign labels, for example, based on subsequent research (e.g., after identifying one group of customers as "young risk takers").

Unsupervised learning is the great promise of the future. It shouts that computers could someday learn on their own in a true robotic sense. Currently, this learning method is limited to networks known as selforganizing maps. These kinds of networks are not in widespread use. They are basically an academic novelty. Yet, they have shown they can provide a solution in a few instances, proving that their promise is not groundless. They have been proven to be more effective than many algorithmic techniques for numerical aerodynamic flow calculations. They are also being used in the lab where they are split into a front-end network that recognizes short, phoneme-like fragments of speech which are then passed on to a backend network. The second artificial network recognizes these strings of fragments as words.

This promising field of unsupervised learning is sometimes called selfsupervised learning. These networks use no external influences to adjust their weights. Instead, they internally monitor their performance. These networks look for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. This information is built into the network topology and learning rules.

An unsupervised learning algorithm might emphasize cooperation among clusters of processing elements. In such a scheme, the clusters would work together. If some external input activated any node in the cluster, the cluster's activity as a whole could be increased. Likewise, if external input to nodes in the cluster was decreased, that could have an inhibitory effect on the entire cluster.

Competition between processing elements could also form a basis for learning. Training of competitive clusters could amplify the responses of specific groups to specific stimuli. As such, it would associate those groups with each other and with a specific appropriate response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated.

At the present state of the art, unsupervised learning is not well understood and is still the subject of research. This research is currently of interest to the government because military situations often do not have a data set available to train a network until a conflict arises.

3.5.2. Supervised Learning in Neural Network

This method works on reinforcement from the outside. The connections among the neurons in the hidden layer are randomly arranged, then reshuffled as the network is told how close it is to solving the problem. Reinforcement learning is also called supervised learning, because it requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results.

The term "supervised" learning is usually applied to cases in which a particular classification is already observed and recorded in a training sample, and you want to build a model to predict those classifications (in a new testing sample). For example, you may have a data set that contains information about who from among a list of customers targeted for a special promotion responded to that offer. The purpose of the classification analysis would be to build a model to predict who (from a different list of new potential customers) is likely to respond to the same (or a similar) offer in the future.

The vast majority of artificial neural network solutions have been trained with supervision. In this mode, the actual output of a neural network is compared to the desired output. Weights, which are usually randomly set to begin with, are then adjusted by the network so that the next iteration, or cycle, will produce a closer match between the desired and the actual output. The learning method tries to minimize the current errors of all processing elements. This global error reduction is created over time by continuously modifying the input weights until an acceptable network accuracy is reached.

With supervised learning, the artificial neural network must be trained before it becomes useful. Training consists of presenting input and output data to the network. This data is often referred to as the training set. That is, for each input set provided to the system, the corresponding desired output set is provided as well. In most applications, actual data must be used. This training phase can consume a lot of time. In prototype systems, with inadequate processing power, learning can take weeks. This training is considered complete when the neural network reaches an user defined performance level. This level signifies that the network has achieved the desired statistical accuracy as it produces the required outputs for a given sequence of inputs. When no further learning is necessary, the weights are typically frozen for the application. Some network types allow continual training, at a much slower rate, while in operation. This helps a network to adapt to gradually changing conditions.

Training sets need to be fairly large to contain all the needed information if the network is to learn the features and relationships that are important. Not only do the sets have to be large but the training sessions must include a wide variety of data. If the network is trained just one example at a time, all the weights set so meticulously for one fact could be drastically altered in learning the next fact. The previous facts could be forgotten in learning something new. As a result, the system has to learn everything together, finding the best weight settings for the total set of facts.

For example, in teaching a system to recognize pixel patterns for the ten digits, if there were twenty examples of each digit, all the examples of the digit seven should not be presented at the same time.

How the input and output data is represented, or encoded, is a major component to successfully instructing a network. Artificial networks only deal with numeric input data. Therefore, the raw data must often be converted from the external environment. Additionally, it is usually necessary to scale the data, or normalize it to the network's paradigm. This pre-processing of real-world stimuli, be they cameras or sensors, into machine readable format is already common for standard computers. Many conditioning techniques which directly apply to artificial neural network implementations are readily available. It is then up to the network designer to find the best data format and matching network architecture for a given application.

After a supervised network performs well on the training data, then it is important to see what it can do with data it has not seen before. If a system does not give reasonable outputs for this test set, the training period is not over. Indeed, this testing is critical to insure that the network has not simply memorized a given set of data but has learned the general patterns involved within an application.

Both unsupervised and reinforcement suffers from relative slowness and inefficiency relying on a random shuffling to find the proper connection weights.

3.5.3 Back-propagation Training Algorithm

The back-propagation algorithm is perhaps the most widely used training algorithm for multi-layered feed-forward networks. However, many people find it quite difficult to construct multilayer feed-forward networks and training algorithms, whether it is because of the difficulty of the math or the difficulty involved with the actual coding of the network and training algorithm.

Multilayer feed-forward networks normally consist of three or four layers, there is always one input layer and one output layer and usually one hidden layer, although in

some classification problems two hidden layers may be necessary, this case is rare however. The term input layer neurons are a misnomer, no sigmoid unit is applied to the value of each of these neurons. Their raw values are fed into the input layer. Once the neurons for the hidden layer are computed, their activations are then fed to the next layer, until all the activations finally reach the output layer, in which each output layer neuron is associated with a specific classification category. In a fully connected multilayer feed-forward network (See Figure 3.11), each neuron in one layer is connected by a weight to every neuron in the previous layer. A bias is also associated with each of these weighted sums. Thus in computing the value of each neuron in the hidden and output layers one must first take the sum of the weighted sums and the bias and then apply f(sum) (the sigmoid function) to calculate the neuron's activation .

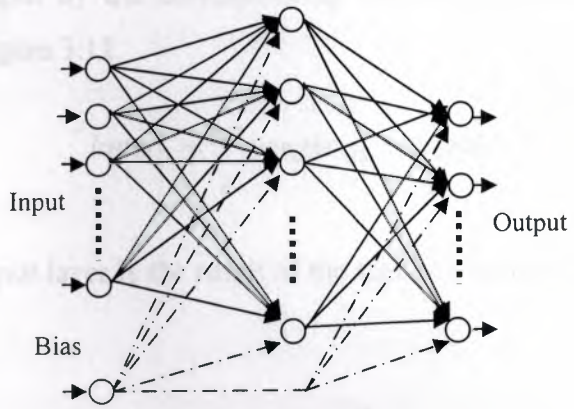


Figure 3.11 Multilayer Feed-Forward Network

3.5.3.1 Feed forward Phase

The feed-forward phase can be described through three steps: input (I), hidden (H), and output layer (O).

- Input layer (I): The output of the input layer is equal to the input of the input layer as shown in Figure 3.11.

$$Output_I = Input_I$$

- Hidden Layer (H): The input of the hidden layer is equal to the sum of multiplying all the input layer output by the corresponding weight that connect between the two layers as shown in Figure 3.11.

$$Input_H = \sum_i wieght_{IHi} * output_I$$

The output of the hidden layer is the result of the sigmoid transfer function of the hidden layer input.

$$Output_H = \frac{1}{1 + e^{-Input_H}}$$

- Output layer (O): The input of the Output layer is equal to the sum of multiplying all the hidden layer output by the corresponding weight that connect between the two layers as shown in Figure 3.11.

$$Input_O = \sum_j wieght_{HOj} * output_H$$

The output of the output layer is the result of the sigmoid transfer function of the output layer input.

$$Output_O \quad \frac{1}{1 + e^{-Input_O}}$$

3.5.3.2 Backpropagation Phase

After the feed-forward phase the output of the output layer ($Output_O$), is compared with the target value of the neural net (Target), the result is the network error ($error_O$).

$$error_O = T\arg et - Ouput_O$$

The purpose of the back-propagation training is to minimize the error of all training pattern by adjusting the weights values, the new value of the hidden-output layer weight is updated according to the following equation.

Generally, several factors besides time have to be considered when discussing the off-line training task, which is often described as "tiresome." Network complexity, size, paradigm selection, architecture, type of learning rule or rules employed, and desired accuracy must all be considered. These factors play a significant role in determining how long it will take to train a network. Changing any one of these factors may either extend the training time to an unreasonable length or even result in an unacceptable accuracy.

Most learning functions have some provision for a learning rate, or learning constant. Usually this term is positive and between zero and one. If the learning rate is greater than one, it is easy for the learning algorithm to overshoot in correcting the weights, and the network will oscillate. Small values of the learning rate will not correct the current error as quickly, but if small steps are taken in correcting errors, there is a good chance of arriving at the best minimum convergence.

3.5.5. Learning laws

There are a variety of learning laws, which are in common use. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sorts of variation of the best-known and oldest learning law, Hebb's Rule. Man's understanding of how neural processing actually works is very limited. Some researchers have the modeling of biological learning as their main objective.

Others are experimenting with adaptations of their perceptions of how nature handles learning. Either way, man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplification represented by the learning laws currently developed. Research into different learning functions continues as new ideas routinely show up in trade publications etc. A few of the major laws are given as examples below.

3.5.5.1.Hebb's Rule

The first and the best known learning rule was introduced by Donald Hebb. The Hebbian Learning Rule is a learning rule that specifies how much the weight of the connection between two units should be increased or decreased in proportion to the product of their activation. The rule builds on Hebbs's 1949 learning rule, which, states that the connections between two neurons might be strengthened if the neurons fire simultaneously.

The Hebbian Rule works well as long as all the input patterns are orthogonal or uncorrelated. The requirement of orthogonal places serious limitations on the Hebbian Learning Rule.

3.5.5.2 Hopfield Law

This law is similar to Hebb's Rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "if the desired output and the input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate." (Most learning functions have some provision for a learning rate, or a learning constant. Usually this term is positive and between zero and one.)

3.5.5.3 The Delta Rule

The Delta Rule is a further variation of Hebb's Rule, and it is one of the most commonly used. This rule is based on the idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a neuron. This rule changes the connection weights in the way that minimizes the mean squared error of the network. The error is back propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. The network type called Feed forward, Back-propagation derives its name from this method of computing the error term.This

rule is also referred to as the Windrow-Hoff Learning Rule and the Least Mean Square Learning Rule.

3.5.5.4 The Gradient Descent Rule:

This rule is similar to the Delta Rule in that the derivative of the transfer function is still used to modify the delta error before it is applied to the connection weights. Here, however, an additional proportional constant tied to the learning rate is appended to the final modifying factor acting upon the weight. This rule is commonly used, even though it converges to a point of stability very slowly.

It has been shown that different learning rates for different layers of a network help the learning process converge faster. In these tests, the learning rates for those layers close to the output were set lower than those layers near the input. This is especially important for applications where the input data is not derived from a strong underlying model.

3.5.5.5 Kohonen's Learning Law

This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the neurons compete for the opportunity to learn, or to update their weights. The processing neuron with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbours. Only the winner is permitted output, and only the winner plus its neighbours are allowed to update their connection weights.

The Kohonen rule does not require desired output. Therefore it is implemented in the unsupervised methods of learning. Kohonen has used this rule combined with the on-centre/off-surround intra- layer connection (discussed earlier) to create the self-organizing neural network, which has an unsupervised learning method.

3.6 Summary

In this chapter, the structures of the neural network are given, the mathematical model of NN and their adjustable parameters starting from single neuron with many inputs to multilayer neural network, and then most common used activation functions such as Log-Sigmoid, Identity function, Binary step function are described.

Training of Neural Network is discussed in detail. Supervised, unsupervised and error backpropagation methods and differences among them are considered.

The brief introduction to learning algorithms, such as Hebb's Rule, Hopfield Law, The Delta Rule are given.

CHAPTER FOUR. DEVELOPMENT OF NEURAL CONTROL SYSTEM

4.1 Overview

The traditional algorithms and artificial intelligence methods do not always adequately describe some processes for complex objects that are characterized nonlinearities. An effective method for development of control system is using the artificial intelligence ideas, such as neural technology for developing the control systems. NN allows to improve the quality of the systems by paralleling computational processes, improve flexibility of systems by means of learning and adaptation abilities.

The use of neural networks control to solve the problem of controlling nonlinear dynamic systems has received attention from many researchers due to their potential in dealing with complex and nonlinear mappings. Neural networks can map complex relations without an explicit set of rules and has a very good learning ability.

The complexity of a number of technological processes and the pressing regime of their functioning require the use of more qualitative control algorithms for regime parameters that provide possibility of learning and adaptation to changes in the environment. However the algorithms developing on the base of traditional approach are complex and their implementation is difficult.

One of effective method for development of control system is using the artificial intelligence ideas. However, the traditional algorithms and artificial intelligence methods do not always adequately describe some processes for complex objects.

In this condition it is advisable to use neural technology for developing of the control systems. Using it allows to improve the quality of systems by paralleling computational processes. The ability of learning and adaptation improve flexibility of the systems.

In this chapter, the development of direct and inverse controllers based on neural network is considered.

4.2 The Development of Neural Control Systems

4.2.1 Supervisory Control

The neural controller in the system is utilized as an inverse system model as shown in Fig. 4.1. The inverse model is simply cascaded with the controlled system such that the system produces an identity mapping between the desired response (i.e., the network input r) and controlled system output y. This control scheme is very common in robotics applications and is appropriate for guidance law and autopilot designs. Success with this model clearly depends on the fidelity of the inverse model used as the controller.

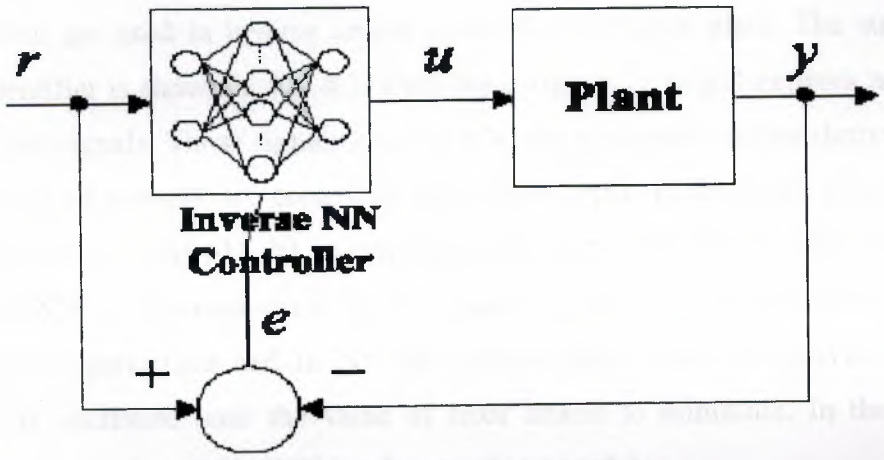


Figure 4.1. Supervisory control scheme

4.2.2 Synthesis of direct neural controller

The structure of inverse neural controller is shown in figure 4.2. The inputs for the neural controller are one and two- step delayed outputs of the plant, one step delayed output of neural controller and the reference signal. These signals are given to the neural network input, after processing in neural network block the output control signal of network is entered to the plant input. Plant outputs are input for the controller.

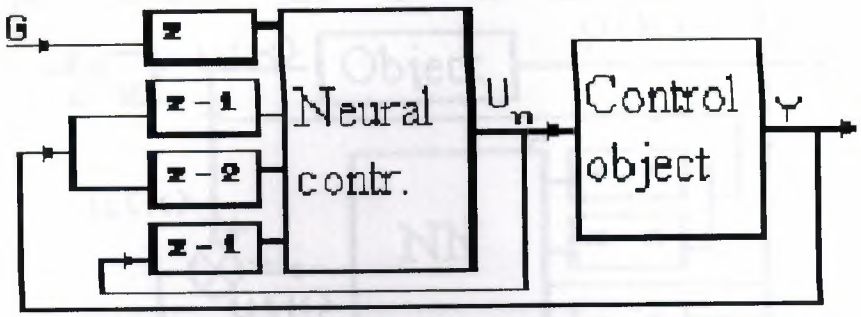


Figure 4.2 Structure of inverse controller

The main problem here is constructing neural controller. For this reason the result of inverse identification is used. The weight coefficients obtained from inverse identification are used in inverse neural controller to control plant. The structure of inverse identifier is shown in fig. 4.3. Here input signals of neural network are control object output signals. Those signals enter to NN, are processed and the derived signals on the output of network are compared with object input. In the result of comparison the value of error $E=U(k)-U_N(k)$ is calculated. This error corrects the value of synaptic weights of NN to minimise error. In the result of learning the correction of weight coefficients is performed and in NN the inverse plant model is derived. Learning processes is continued until the value of error attains to minimum. In the result of learning the derived model on NN is taken as object model.

For learning of NN the 'backpropagation' algorithms is used. In the NN the following activation function is used.

$$Y_N=X/(A+|X|) \qquad (4.1)$$

Using feedforward neural network the construction of the neural controllers is described in [20].

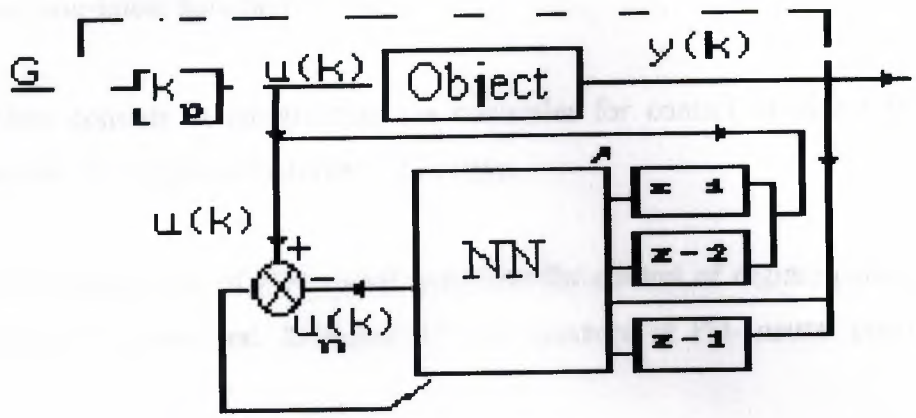


Figure 4.3 Structure of inverse identification

The program performing inverse identification processes and controlling object is developed. The system is implemented using Turbo Pascal and a computer IBM PC/AT In fig. 4.4 the simulation result that demonstrates time response of the system with inverse neural controller is shown. Although the inverse identification of the object and use of their results in the inverse neural controller require certain time.

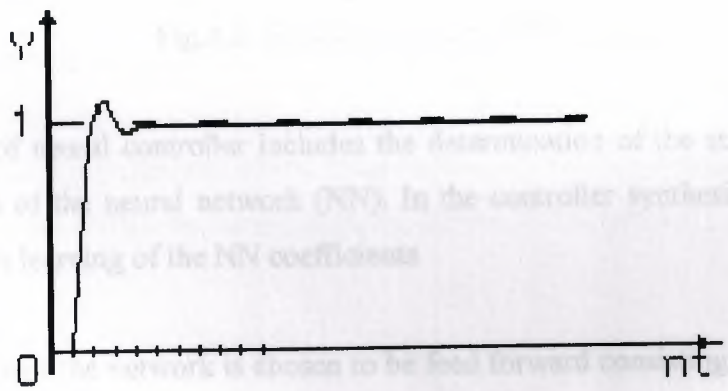


Figure 4.4. Time response of control system

4.3 Development Neural Control System

Assume that control object is described by the following differential equation

$$\sum_{i=1}^{n} a_{n-i} y^{(i)}(t) + c\varphi(y(t)) = \sum_{j=1}^{m} b_{m-j} u^{(j)}(t) \qquad (4.2)$$

Where a_i ($i=\overline{1,n}$) and b_j ($j=\overline{1,m}$) are unknown parameters of control object, d is delay; c is unknown non-linear parameter, $m<n$.

The problem consists in constructing the controller for control of object (4.2) that would provide the target characteristic of system.

At first the development of PD- neural controller for control of regime parameters of control object is considered. In figure 4.5 the structure of PD- neural controller is shown.

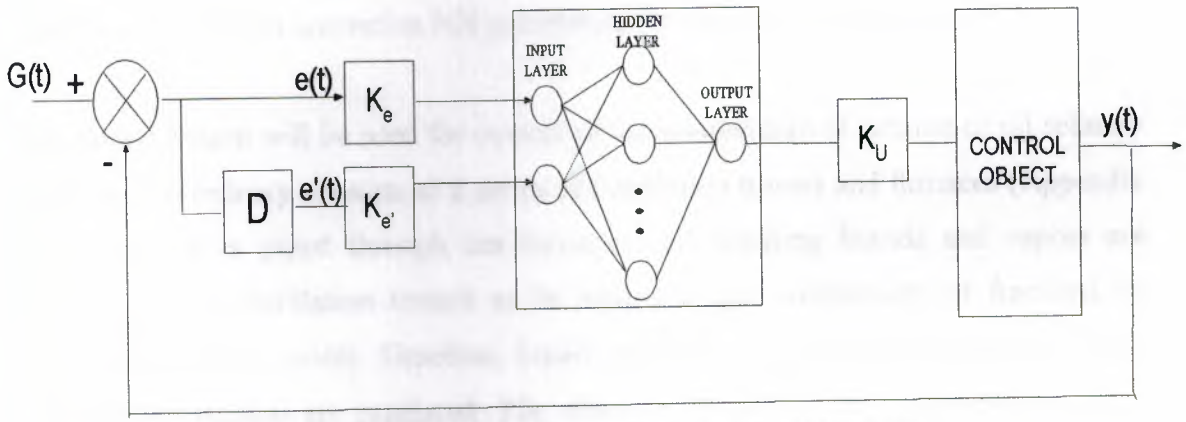


Fig.4.5. Structure of neural PD- controller.

The synthesis of neural controller includes the determination of the scale coefficients and parameters of the neural network (NN). In the controller synthesis processes the main problem is learning of the NN coefficients.

The architecture of the network is chosen to be feed forward consisting of three layers: input, hidden and output layer. The problem of control system synthesis on the base of NN is the following:

Assume there is target behavior for the constructed control system. It is necessary to determine the values of parameters- weight matrix w_{ij} and scale coefficients using of which in control system for object (4.2) would allow achieving time response, which provides target step response of the system.

EAST UNIVERSITY LIBRARY

The input signals error e, and error derivative e' after scaling with coefficients k_e, $\mathbf{k}_e$, are entered to neural network. The functioning of neural network is performed by using activation function.

$$U=Y/(A+|Y|). \text{ Here } Y=XW. \qquad (4.3)$$

For synthesis of neural controller the NN learning is performed by using 'back propagation' algorithm. The NN learning is performed in the closed control system, i.e. for learning NN error between target characteristic of control system and current output value of implemented system (output of control object) $\Delta(y, t)=k_e(g(t)-y(t))$ is used. That error is used for correction NN parameters for adjusting of controller.

Developed Neural will be used for control of top temperature of column of oil refinery plant. An oil refinery consists of a series of distillation towers and furnaces (Appendix D). Crude oil is piped through hot furnaces and resulting liquids and vapors are discharged into distillation towers to be separated into components or fractions by weight and boiling point. Gasoline, liquid petroleum gas, kerosene, diesel oil, and intermediate streams are produced. The effective control of the parameters such as temperature, pressure in the column define the quality of these products. In the thesis the controlling top temperature of of columun is considered. The simulation of control system for control top temperature have been carried out. For the simulation the models of control object are chosen by using following differential equations:

$$a_0y^{(2)}(t)+a_1y^{(1)}(t)+a_2y(t)=b_0u(t-d) \qquad (4.4)$$

here y(t)- regulation parameter of object, u(t)- neural controller's output, where a_0= 0.072 min^2 , a_1=0.56 min, a_2=1, b_0=60 °C/(kgf/cm^2), d=0.15 min is delay.

The neural controllers development for given control objects are performed. In the result of learning corresponding values of neural network coefficients are determined.

The Program that simulates neural control system is given in appendix A. At first stage using generalization button the initial values of weight coefficients of neural control

randomly generalized. Second picture of appendix A demonstrate value of weight coefficients.

Using learning button the learning of neural control is carried out (Appendix B). In figure 4.6 learning progress of neural control system is given. Using "saving parameters" button, the learned parameters values of neural control are stored in file. Test button stage is used for control of dynamic plant.

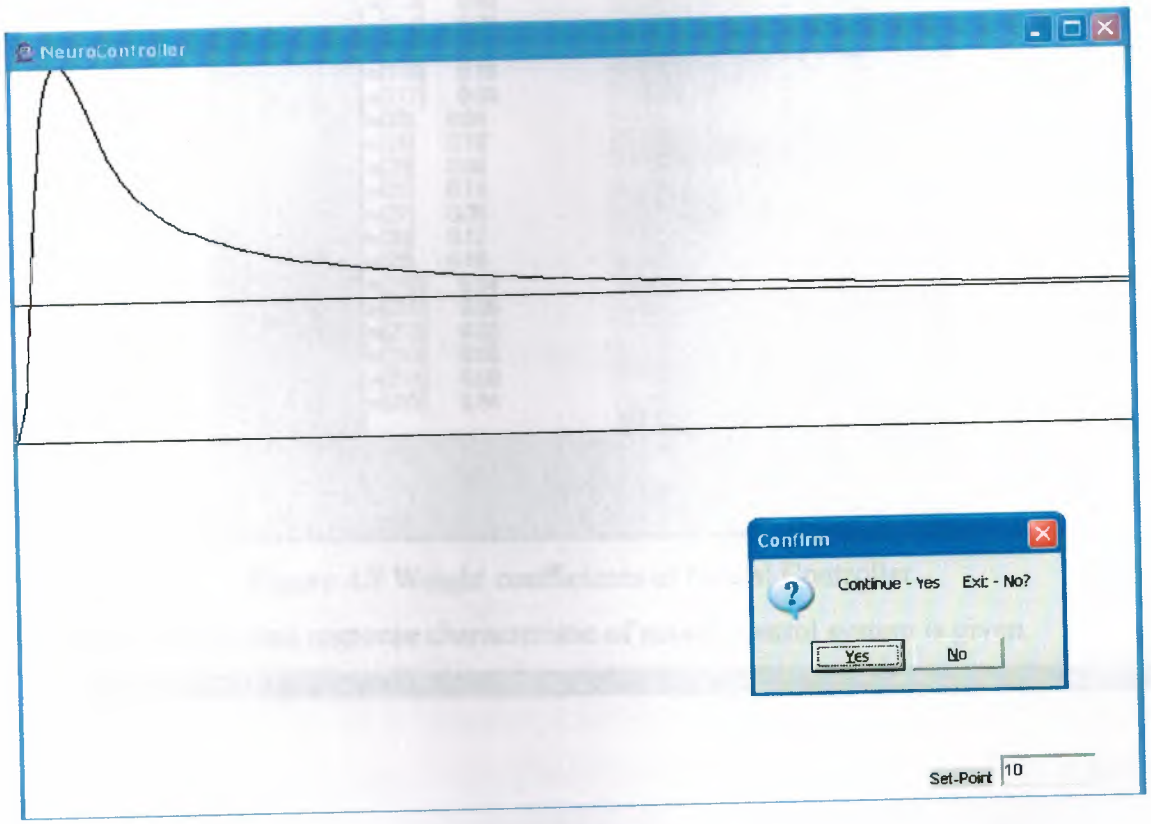


Figure 4.6 Learning process of neural control system

After learning using founded weight coefficients of neural controller the online control of plant (4.4) have been carried out. In figure 4.7 the values of founded weight coefficients are shown.

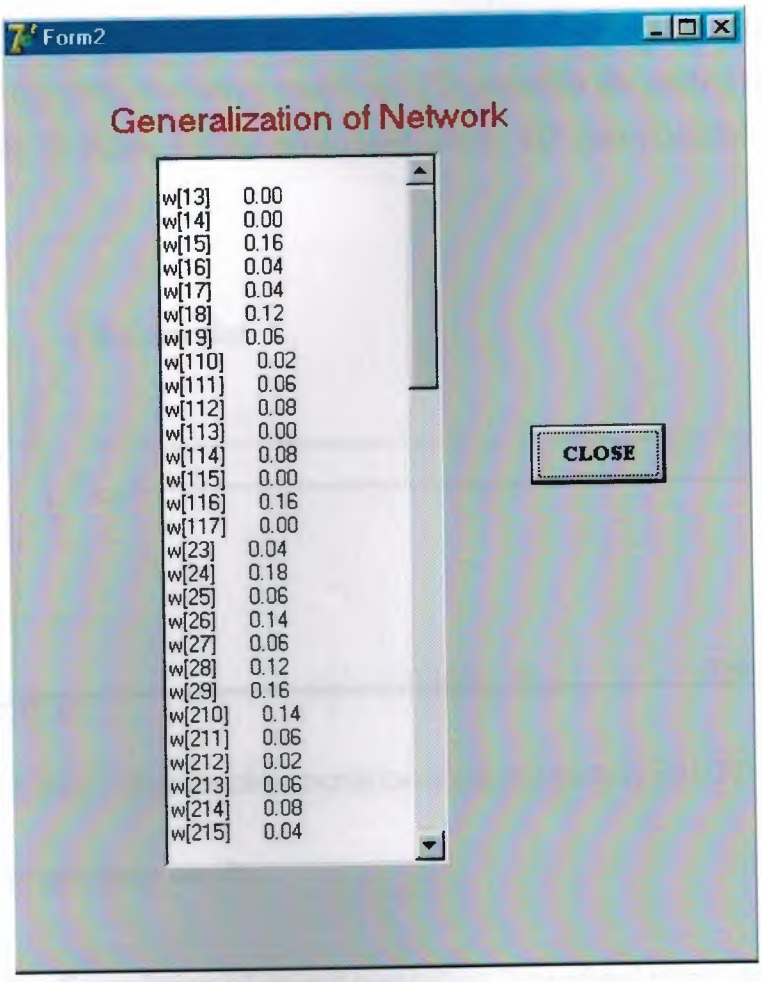


Figure 4.7 Weight coefficients of Neural Controller.

In figure 4.8 the time response characteristic of neural control system is given.

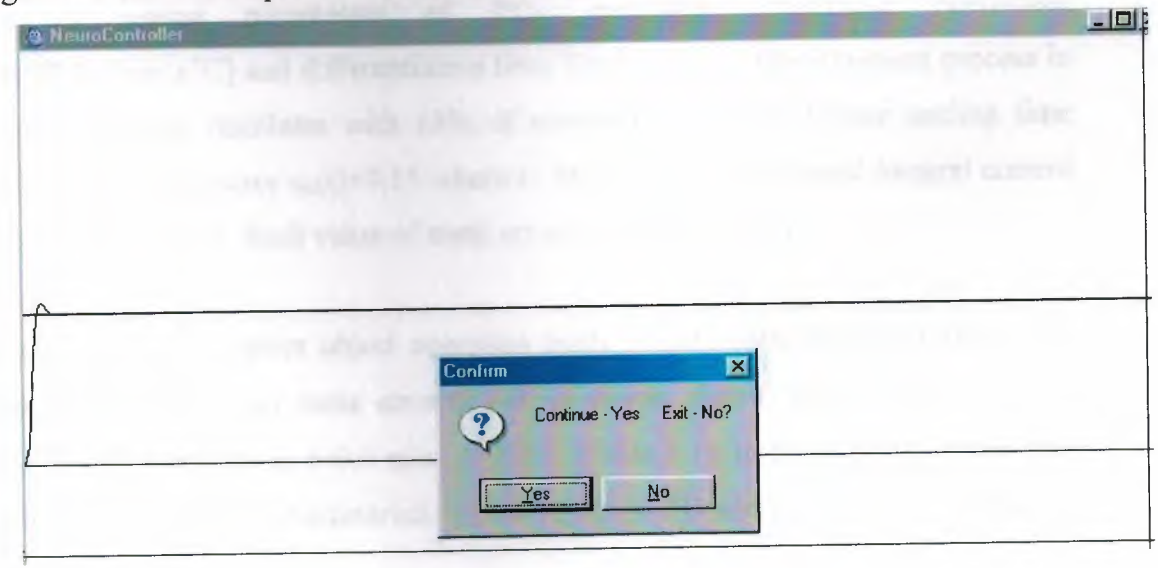


Figure 4.8 Automatic control systems with neural controller

The graphical presentation of control system with traditional PD controller for plant (4.4) is taken from [20]. Also using traditional PD controller the control of plant (4.4) is carried out [20]. In figure 4.9 the time responses of PD controller for control object (4.4) is shown.

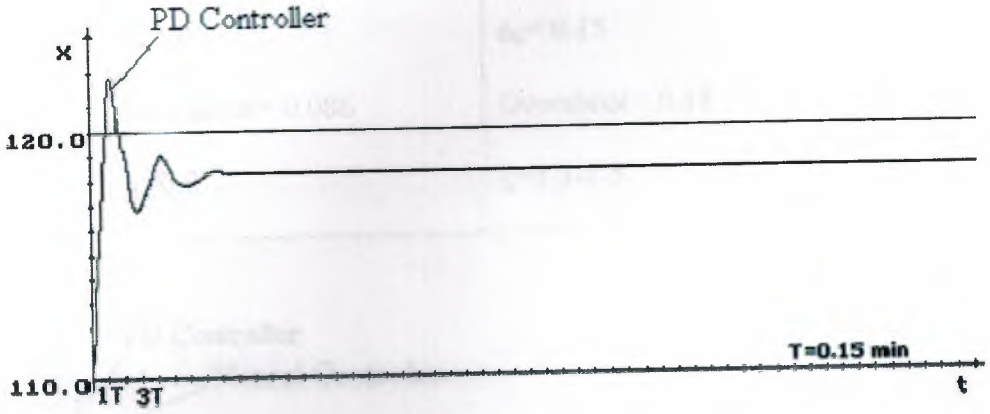


Figure 4.9 Time response characteristics of control system with PD controller

4.4. Analysis of obtained results

Then the results of simulation of neural controllers for technological processes control are compared with simulation results of the traditional PD controller. When optimal value of tuning parameters of PD- controller amplifying coefficient kp=0.08[(kgf/cm^2)/°C] and differentiation time Td=0.15 min., then transient process in the control system oscillates with 18% of transient overshoot. Where settling time t=1.3-1.5 min, static error $\varepsilon_{st}(t)$=0.15 where t> 9t, and value of squared integral control quality index J=276.4. Such value of static error is not satisfactory.

One can see from transient object operation mode of automatic control system with neural PD-controller that static error ($\varepsilon_{st}\approx 0$) is almost absent, transient overshoot is almost 8.6%, settling time t=0.9 min., J=152.7 (Table 4.1). In figure 4.10 comparative +result of time response characteristic of conventional PD and neural PD controllers is given.

Table 4.1 Comparison Results of PD and Neural PD Controllers

Neural Controller	PD Controller
J=152.7	J=276.4
e_{st}= 0	e_{st}= 0.15
Overshoot= 0.086	Overshoot= 0.18
t_s=0.9	t_s=1.3-1.5

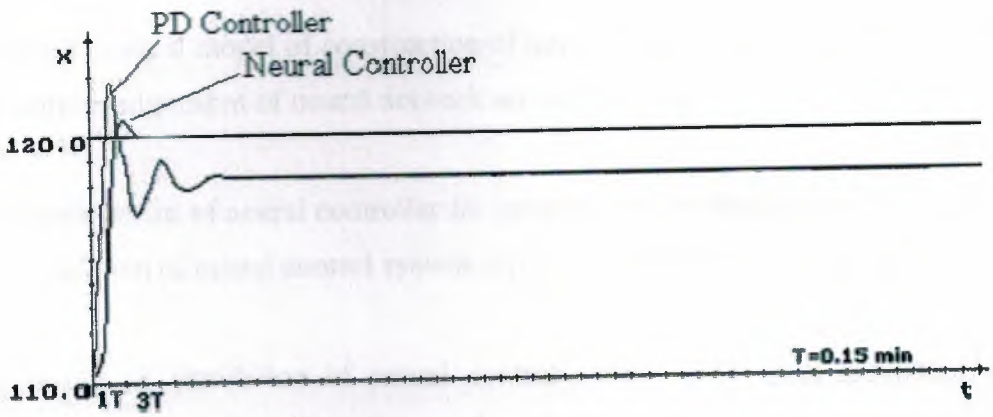


Figure 4.10 Comparison between Traditional PD- and Neural Controller

The graphical presentation of control system with traditional PD controller for plant (4.4) is taken from [20]. Comparison results show that neural control system is more efficient.

4.5 Summary

The development of neural control system has been performed. Using Delphi programming the modeling of neural controller is carried out. The simulation result of implemented system is compared with result of traditional system. The obtained results demonstrate the efficiency of neural technology in control system development.

CONCLUSION

Analyses of technological processes show that the traditional controllers do not give desired results during controlling complicated process characterizing non-linearity. For these processes it is advisable to use controller based on neural network.

To develop neural controller for these processes in the thesis the following have been performed.

The structures of neural controllers for technological process control and the functions of their main blocks are given.

The mathematical model of construction of neural controller is presented. The structure and learning algorithm of neural network are represented.

The development of neural controller for control of technological process is carried out. The simulation of neural control system is performed. The result of simulation is given

The results of simulation of neural control system have been compared with the simulation results of traditional control system. Comparative results demonstrate the improvement of accuracy of control of system based on neural controller.

REFERENCE

[1] Narendra, K. S. and K. Parthasarthy (1990) Identification and control of dynamical using neural networks. *IEEE Trans. Neural Networks*, 1(1), 4-27.

[2] Hunt, K. J. and D. Sbarbaro-Hofer (1991) Neural networks for nonlinear internal model control. *IEE Proc. Pt. D*, 138(5), 431-438.

[3] Silvia Ferrari, Algebraic and Adaptive Learning in Neural Control Systems, August, 2002, Princeton, New Jersey

[4] Rahib Abiyev, Recurrent Neural Network Systems for Control of Dynamic Plants Near East University, Department of Computer Engineering, Mersin-10 Turkey.

[5] Martin T. Hagan, An Introduction to The Use of Neural Networks in Control Systems School of Electrical & Computer Engineering, Oklahoma State University, Stillwater, Oklahoma, 74075, USA

[6] Cairo L. Nascimento,Artificial Neural Networks in Control and Optimization, Control Systems Centre, University of Manchester Institute of Science and Technology, February, 1994 Manchester

[7] Daniel Eggert Neural Network Control Technical University of Denmark Informatics and Mathematical Modelling Lyngby, Denmark February 2003

[8] Jens Haecker and Stephan Rudolph Institute for Statics and Dynamics of Aerospace Structures Universität Stuttgart, Pfaffenwaldring 27, D-70569 Stuttgart, Germany

[9] K. Kantapanit, C. Treesatayapun, W. Wiriyasuttiwong Adaptive Neural Network Control with Plant Information Feedbac. ChiangMai University, Chiang Mai, Thailand

[10] U. Halici, K. Leblebicioglu, Department of Electrical and Electronics Engineering, Recent Advances in Neural Network Applications in process control, Middle East University, 06531, Ankara, Turkey

[11] Ömer CİVALEK, Mehmet ÜLKER, Artificial Neural Networks Approach to The Non-Linear Analysis of Rectangular Plates,IMO Teknik Dergi, 2004 3171-3190, Yazi 213

[12] Rahip Abiyev, The Structure of Intellectual Neural Control Systems for Technological Process, Nicosia 2002

[13] Lloyd Greenwald and Donovan Artz, Teaching Artificial Intelligence with Low-Cost Robots Department of Computer Science, Drexel University, Philadelphia

[14] Michael Nikolaoul and Vijaykumar Hanagandi, Control of Nonlinear Dynamical Systems Modelled by Recurrent Neural Networks, Department of Chemical Engineering, Texas A&M University

[15] Tao ZHANG, Ph.D. Intelligent Systems Research Division, National Institute of Informatics 2-1-2, Hitotsubashi, Chiyoda-Ku, Tokyo 101-8430 JAPAN

[16] *S*. Rahman, Short-Term Load Forecasting with local ann predictors. Center for Energy and the Global Environment Virginia Polytechnic Institute and State University Blacksburg, USA

[17] Control in an Information Rich World, Report of the Panel on Future Directions in Control, Dynamics, and Systems 30 June 2002

[18] Jay T. Moody, Visualizing Speech with a Recurrent Neural Network Trained on Human Acoustic-Articulatory Data, University Of California, San Diego, 1999

[19] Sargur Srihari, Artificial Neural Network,Department of Computer Science and Engineering, University at Buffalo,2005

[20] R.H.Abiyev Neural Learning Systems for Technological Processes Control, Azerbaijan State Oil Academy, Dep. Automatic Control System Baku,Azerbaijan,1994

[21] R.H.Abiyev, R.A.Aliev, R.R.Aliev. Synthesis of automatic control system by learned neural network based fuzzy controller// News of Academy of Sciences, Journal of Tech. Cybernetics 2: 192-197, Moscow, 1994 (in English and Russian).

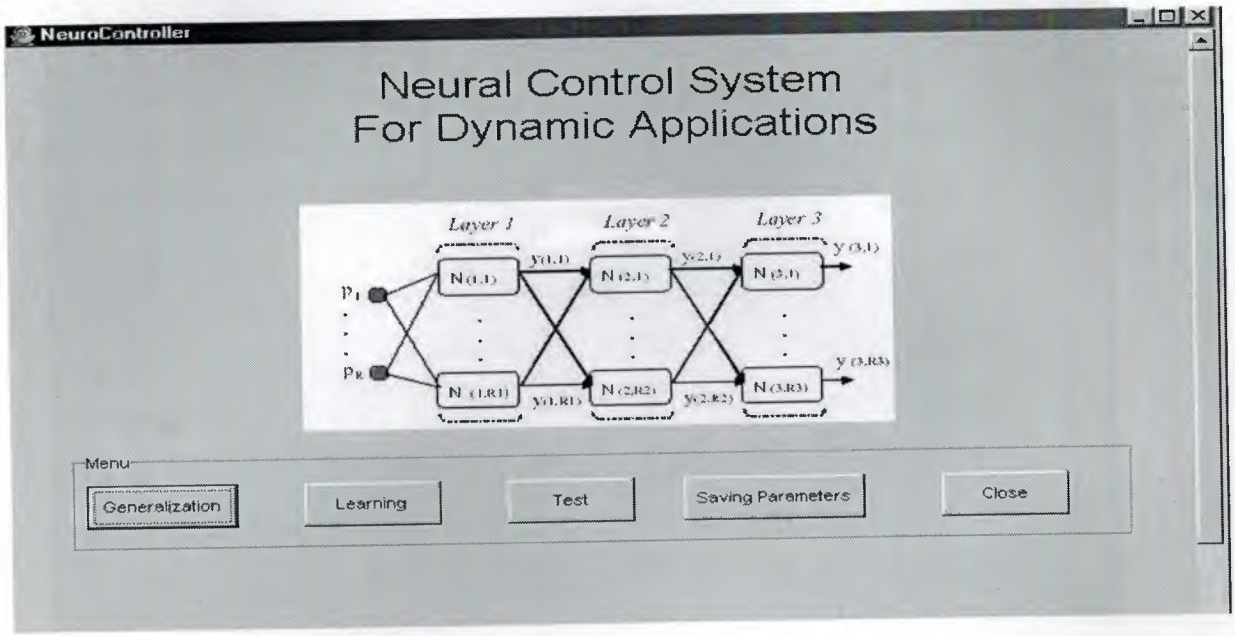
[22] R.H.Abiyev, K.W.Bonfig, F.T.Aliev. Controller based on fuzzy neural network for control of technological process.// International conferences on Application of Fuzzy Systems and Softcomputing, ICAFS-96, Siegen, Germany, June 25-27, 1996,pp.295-298.

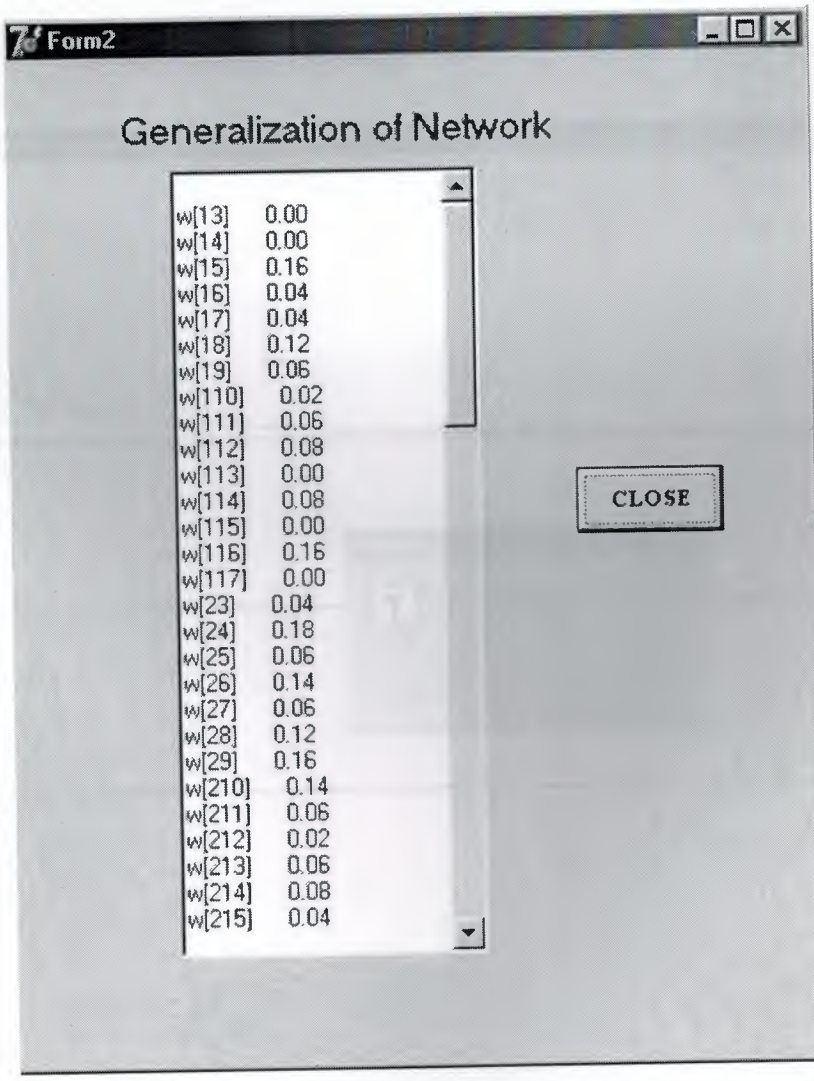
[23] F.T.Aliew, R.R.Aliev, W.steinmann, I.H.Murator, R.H.Abiyev. Neural Learning System For Technological processes Control // International Conference on Application of Fuzzy System and Soft Computing, ICAFS-98,Wiesbaden,Germany

[24] R.H.Abiyev. Controllers based on neural networks// Uchenie zapiski, AzGNA, 1995, 147-152pp.(Russian)

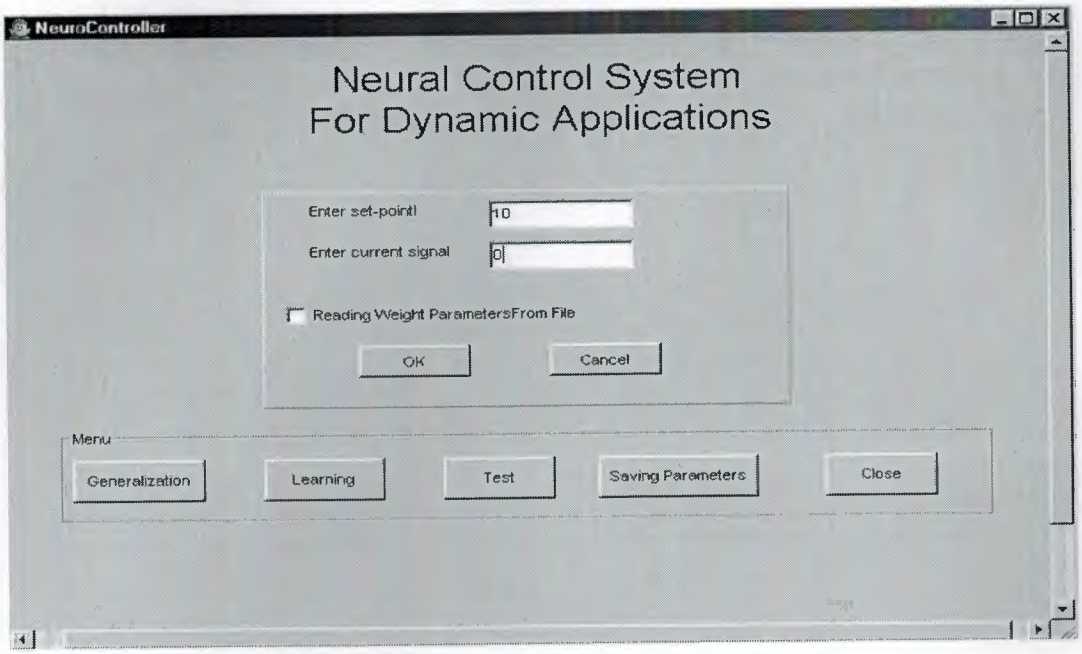
[25] Narendra, K. S., "Adaptive Control of Dynamical Systems Using Neural Networks," *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, White D. A., and Sofge D. A., ed., pp. 141-184, 1992.

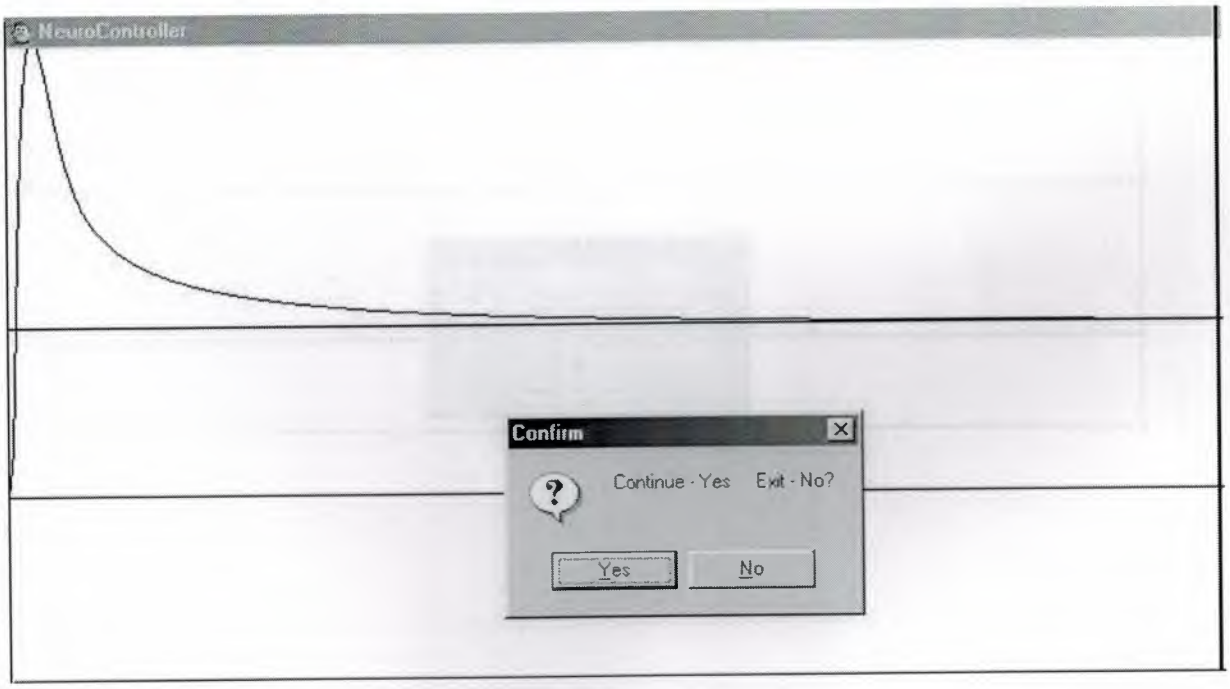
APPENDIX A

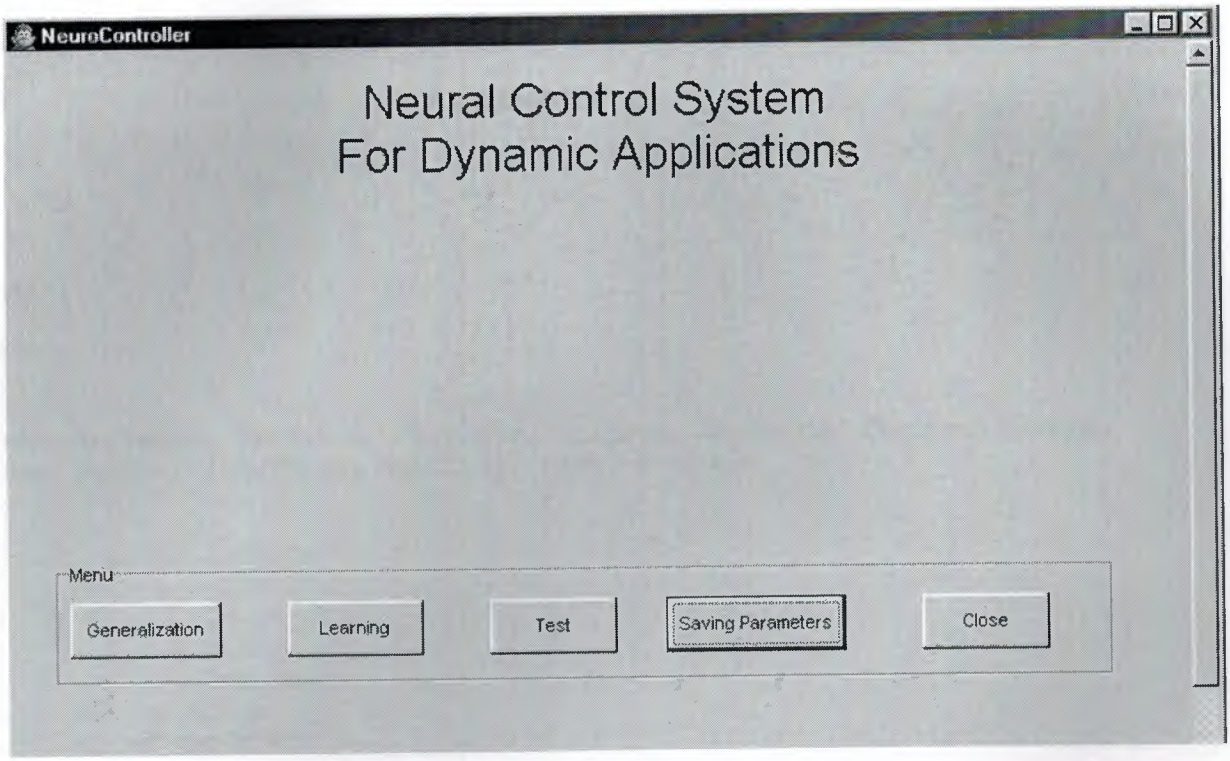

NeuroController
Neural Control System
For Dynamic Applications
Layer 1
Layer 2
Layer 3
Menu
Generalization
Learning
Test
Saving Parameters
Close

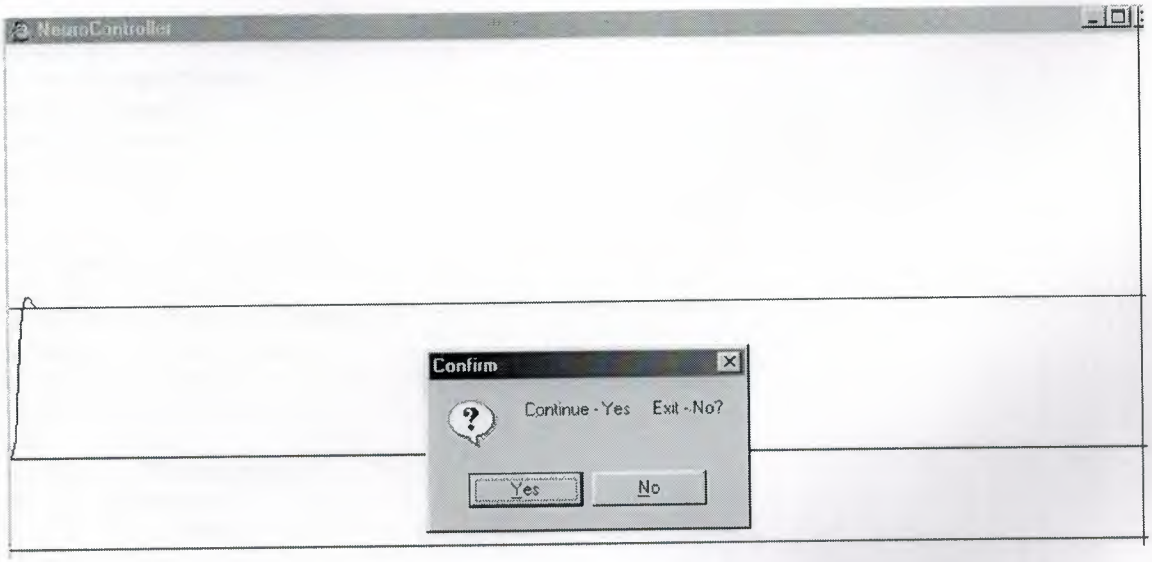

Form2
Generalization of Network
w[13] 0.00
w[14] 0.00
w[15] 0.16
w[16] 0.04
w[17] 0.04
w[18] 0.12
w[19] 0.06
w[110] 0.02
w[111] 0.06
w[112] 0.08
w[113] 0.00
w[114] 0.08
w[115] 0.00
w[116] 0.16
w[117] 0.00
w[23] 0.04
w[24] 0.18
w[25] 0.06
w[26] 0.14
w[27] 0.06
w[28] 0.12
w[29] 0.16
w[210] 0.14
w[211] 0.06
w[212] 0.02
w[213] 0.06
w[214] 0.08
w[215] 0.04
CLOSE

APPENDIX B


NeuroController
Neural Control System
For Dynamic Applications
Enter set-point!
10
Enter current signal
0
Reading Weight ParametersFrom File
OK
Cancel
Menu
Generalization
Learning
Test
Saving Parameters
Close


NeuroController
Confirm
Continue - Yes Exit - No?
Yes
No


NeuroController
Neural Control System
For Dynamic Applications
Menu
Generalization
Learning
Test
Saving Parameters
Close


NeuroController
Confirm
Continue - Yes Exit - No?
Yes
No

APPENDIX C

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtDlgs, ExtCtrls, Mask, Buttons;

const
n1=2; n2=17; m=18; ll=1000;
kx=10;

type
u=array [1..n2,1..m]of real;
r=array[1..n1] of real;
q=array[1..m] of real;
c=file of q;
u1=array [0..ll] of real;
uu1=array [0..ll] of integer;
 vv=file of u;
  TForm1 = class(TForm)
    Label1: TLabel;
    Image1: TImage;
    Timer1: TTimer;
    Panel1: TPanel;
    MaskEdit1: TMaskEdit;
    StaticText1: TStaticText;
    MaskEdit2: TMaskEdit;
    StaticText2: TStaticText;
    CheckBox1: TCheckBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Image2: TImage;
    GroupBox1: TGroupBox;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;


    procedure FormCreate(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);


   //procedure Timer1Timer(Sender: TObject);
  private
```

```
      g,x0,rc:real;
      x1,y1,code:integer;
      sh1:string;

    { Private declarations }
  public
    { Public declarations }
  end;
    procedure randw1;
    procedure cons;
var
  Form1: TForm1;
  rc:real;
  w:u;
  i,j,k,l,len:integer;
  x,z:r; xx,xxx:u1;
  y,yp,h,tt,net:q; d2,g,uu,dy,jj:real;
  e:vv; f:c; ch,ch1,ch2,key:char;
  a0,a1,a2,ft,k1,k2,k3,t,aa,aa1,aa2,bb1,bb2,x0:real;

  {graph}
  //x1,y1,x2,y2,gg,kg,kx:integer;
  //ii,y11,x12,y12,y13:uu1;

implementation

uses Unit2;



{$R *.dfm}
 Procedure funk(x:r;var y,net:q);
  var
  i,k:integer;
  BEGIN
  for i:=1 to n1 do  y[i]:=x[i];
  for k:=n1+1 to n2 do
  begin
  net[k]:=0;
  for i:=1 to n1 do
     net[k]:=net[k]+y[i]*w[i,k];
     net[k]:=net[k]+w[n1+1,k]*y[m];
  y[k]:=1/(1+exp(-net[k]));
  end;

  for k:=n2+1 to m do
  begin
  net[k]:=0;
  for i:=n1+1 to n2 do
      net[k]:=net[k]+y[i]*w[i,k];
  y[k]:=1/(1+exp(-net[k]));
  end;

  END;
   Procedure Train;
     var
      i,k,j:integer;
        begin
          for k:=n2+1 to m do
          h[k]:=dy*(1-y[k])*y[k];
```

```
    for i:=n1+1 to n2 do
        for k:=n2+1 to m do
        begin
         w[i,k]:=w[i,k]+d2*h[k]*y[i];
        end;

        for i:=n1+1 to n2 do
         begin
         tt[i]:=0;
         for k:=n2+1 to m do

     h[i]:=(1-y[i])*y[i]*tt[i];
     end;

          for j:=1 to n1 do
          for i:=n1+1 to n2 do
              w[j,i]:=w[j,i]+d2*h[i]*y[j];
     end;

Procedure rrr1;
 var
  i,j:integer;
begin

for i:=1 to n1 do
    for j:=n1+1 to n2 do
    begin
if (w[i,j]>1) or (w[i,j]<-1) then w[i,j]:=random;
end;

    for i:=n1+1 to n2 do
    for j:=n2+1 to m do
    begin
if (w[i,j]>1) or (w[i,j]<-1) then w[i,j]:=random;
end;

end;

procedure randw1;
var
i,j:integer;
begin
for i:=1 to n1 do
for j:=n1+1 to n2 do
w[i,j]:=rc*random(10);

for i:=n1+1 to n2 do
for j:=n2+1 to m do
w[i,j]:=rc*random(10);

end;

procedure cons;
begin
aa1:=(a0*2+a1*t)/(a0+a1*t+a2*t*t);
aa2:=-a0/(a0+a1*t+a2*t*t);
bb1:=(ft*t*t)/(a0+a1*t+a2*t*t);
bb2:=0;
//form2.Memo1.Text:=form2.Memo1.Text+'a1=',aa1:9:6,'a2=','b1',aa2:9:6);
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);

begin
label1.Font.Color:=clnavy;
label1.Font.Size:=20;

label1.Caption:='    Neural Control System '+#13#10+
               '  For Dynamic Applications';

end;

{ procedure TForm1.Timer1Timer(Sender: TObject);
var
x:integer;
begin
x:=0;
while x<3 do
begin
 image1.Picture.LoadFromFile('F:\Neural Network\untitled2.bmp');
 sleep(500);
 image1.Picture.LoadFromFile('F:\Neural Network\untitled.bmp');
 sleep(500);
 x:=x+1; if x=3 then x:=0;
 end;
 end; }

procedure TForm1.Button5Click(Sender: TObject);
begin
Halt;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
i,j:integer;
ch1,ch3:string;
begin
assignfile(e, 'D:\abidin2.dat');
//if ioresult<> 0 then rewrite(e);
rewrite(e);

for i:=1 to n1 do
for j:=n1+1 to n2 do
begin
rc:=0.02;
w[i,j]:=rc*random(10);
str(w[i,j]:10:2,ch1);
form2.Show;
form2.memo1.Text:=form2.memo1.Text+'w['+inttostr(i)
+inttostr(j)+']'+ch1+#13#10;
// listbox1.AddItem(ch1,w[i,j]);
end;
form2.Show;
for i:=n1+1 to n2 do
for j:=n2+1 to m do
begin
w[i,j]:=rc*random(10);
str(w[i,j]:10:2,ch3);
form2.memo1.Text:=form2.memo1.text+'w['+inttostr(i)
+inttostr(j)+']'+ch3+#13#10;
end;
seek(e,filesize(e));
```

```
write(e,w);
closefile(e);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
i,j:integer;
ch1,ch2:string;
begin
panel1.Visible:=true;

val(maskedit1.Text,g,code);
val(maskedit1.Text,x0,code);
rc:=0.02;

end;


procedure TForm1.BitBtn1Click(Sender: TObject);
label 11,13,quit1;
var
i,j:integer;
ch1,ch2:string;
begin
panel1.visible:=false;
Groupbox1.visible:=false;
image1.visible:=false;
image2.Align:=alclient;
x1:=0; y1:=round(image2.Height/2);
if checkbox1.Checked then
begin
  assignfile(e, 'd:\abidin2.dat');
  reset(e);
  read(e,w);
  form2.Show;
  for i:=1 to n1 do
  for j:=n1+1 to n2 do
  begin
   str(w[i,j]:10:2,ch1);

   form2.memo1.Text:=form2.memo1.text+'w['+inttostr(i)
   +inttostr(j)+']'+ch1+#13#10;
  end;
 for i:=n1+1 to n2 do
 for j:=n2+1 to m do
  begin
   str(w[i,j]:10:2,ch2);
   form2.Show;
   form2.memo1.Text:=form2.memo1.text+'w['+inttostr(i)
   +inttostr(j)+']'+ch2+#13#10;
  end;
 end
else
randw1;

d2:=0.0002;
        l:=2; t:=0.12;  aa:=1;
        a0:=0.072; a1:=0.56; a2:=1; ft:=60;
        a0:=0.042; a1:=0.072; a2:=1; ft:=60;
        k1:=1/1; k2:=1/1; k3:=1;  y[m]:=0;
        xx[1-2]:=0;xx[1-1]:=0; xx[1]:=0;
```

```
      xxx[l-2]:=x0; xxx[l-1]:=xxx[l-2]; xxx[l]:=xxx[l-1];
      cons;   jj:=0;
        showmessage('Click a key');
        form2.Visible:=false;
        image2.Visible:=true;
        image2.Canvas.Moveto(x1,y1);
        image2.Canvas.Lineto(x1+image2.Width,y1+Round(kx*g));
        image2.canvas.MoveTo(0,0);


11: while l<=image2.width do begin
      z[1]:=g-xxx[l];
      z[2]:=(xxx[l]-xxx[l-1])/t;  jj:=jj+z[1]*z[1];
      x[1]:=k1*z[1]; x[2]:=k2*z[2];
      funk(x,y,net);
      uu:=k3*y[m];
      l:=l+1;
   xx[l]:=(uu*bb1+aa1*xx[l-1]+aa2*xx[l-2]);
{  writeln(l,'=l',z[1]:8:4,'=z=',z[2]:8:4,' uu=',uu:8:4,' xx=',xx[l]:8:4,'
jj=',jj:8:4); readln;}
   xxx[l]:=x0+xx[l];

   image2.Canvas.LineTo(x1+l,y1+Round(kx*xxx[l]));
    if l=image2.Width-2 then
    begin

     if MessageDlg(' Continue -Enter     Exit-Esc?',
     mtConfirmation, [mbYes, mbNo], 0) = mrYes then
     begin
          //     showmasege('Click a keyyyyyy');
     image2.Canvas.FillRect(Rect(0,0,image2.Width,image2.Height));
     xx[0]:=xx[l-2]; xx[1]:=xx[l-1]; xx[2]:=xx[l];
     xxx[0]:=xxx[l-2]; xxx[1]:=xxx[l-1]; xxx[2]:=xxx[l];
     l:=2;
     image2.Canvas.MoveTo(x1+l,y1+Round(kx*xxx[l]));
     end
     else
     goto quit1;
   end;
 // if len=2 then goto 14;
  if (z[1]>0.004) or (z[1]<-0.004) then
       begin
       dy:=z[1]/k1;
      train;
      rrr1;
      //  if image1.  then goto 14;
      goto 11; end else
      begin

      cons;
      end;


   end;
   quit1: image2.visible:=false;
          form1.visible:=true;
          groupbox1.Visible:=true;
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
form1.Visible:=true;
```

```
cons;   jj:=0;
        edit1.Text:=floattostr(g);

        showmessage('Click a key');
        form2.Visible:=false;
        image2.visible:=true;
        image2.Canvas.FillRect(Rect(0,0,image2.Width,image2.Height));
        image2.Canvas.Moveto(x1,y1);
        image2.Canvas.Lineto(x1+image2.Width,y1);
        image2.canvas.MoveTo(x1,y1-Round(kx*g));
        image2.Canvas.Lineto(x1+image2.Width,y1-Round(kx*g));
        image2.Canvas.Moveto(x1,y1);
11: while l<=image2.width do begin
      z[1]:=g-xxx[l];
      z[2]:=(xxx[l]-xxx[l-1])/t;  jj:=jj+z[1]*z[1];
      x[1]:=k1*z[1]; x[2]:=k2*z[2];
      funk(x,y,net);
      uu:=k3*y[m];
      l:=l+1;
   xx[l]:=(uu*bb1+aa1*xx[l-1]+aa2*xx[l-2]);
{  writeln(l,'=l',z[1]:8:4,'=z=',z[2]:8:4,' uu=',uu:8:4,' xx=',xx[l]:8:4,'
jj=',jj:8:4); readln;}
   xxx[l]:=x0+xx[l];
   z[1]:=(g-xxx[l]);
   image2.Canvas.LineTo(x1+l,y1-Round(kx*xxx[l]));
    if l=image2.Width-2 then
    begin

     if MessageDlg(' Continue - Yes      Exit - No?',
     mtConfirmation, [mbYes, mbNo], 0) = mrYes then
     begin

          //     showmasege('Click a keyyyyyy');
     image2.Canvas.FillRect(Rect(0,0,image2.Width,image2.Height));
     xx[0]:=xx[l-2]; xx[1]:=xx[l-1]; xx[2]:=xx[l];
     xxx[0]:=xxx[l-2]; xxx[1]:=xxx[l-1]; xxx[2]:=xxx[l];
     l:=2;
          image2.Canvas.Moveto(x1,y1);
          image2.Canvas.Lineto(x1+image2.Width,y1);
          image2.canvas.MoveTo(x1,y1-Round(kx*g));
          image2.Canvas.Lineto(x1+image2.Width,y1-Round(kx*g));
     image2.Canvas.MoveTo(x1+l,y1-Round(kx*xxx[l]));
     end
     else
     goto quit1;
   end;
  // if len=2 then goto 14;
   if (z[1]>0.004) or (z[1]<-0.004) then
         begin
         dy:=z[1]/k1;
      train;
      rrr1;
      //  if image1.  then goto 14;
      goto 11; end else
      begin
     {  14: if form1.KeyPress(ord(var key:char)) then
      begin
      Ch := ReadKey;
      if Ch = #59 then
      begin
      Ch := ReadKey;
```

```
    xxx[0]:=xxx[l-2]; xxx[1]:=xxx[l-1]; xxx[2]:=xxx[l];
    l:=2;
    image2.Canvas.MoveTo(x1+l,y1+Round(kx*xxx[l]));
    end
    else goto quit1;
   end;
     begin
    {      Ch := ReadKey;
     { WriteLn(' Enter set-point value ');
          readln(g);                    }
     end;

  end;
 quit1: image2.visible:=false;
        form1.visible:=true;
        groupbox1.Visible:=true;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
assignfile(e, 'd:\abidin2.dat');
rewrite(e);
write(e,w);
closefile(e);
end;

end.
```

APPENDIX D

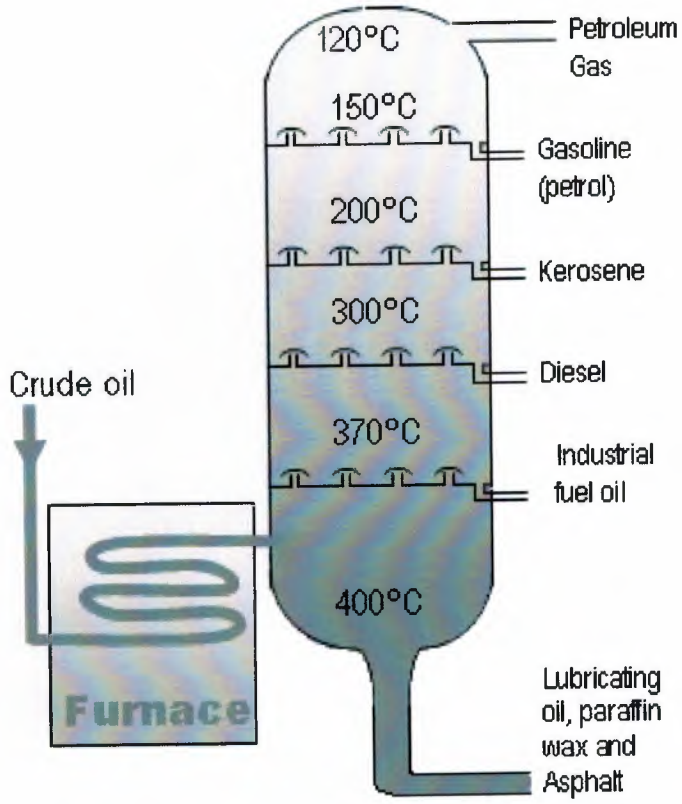


Figure D1 Structure of Oil Refinery