NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

INTELLECTUAL CONTROL SYSTEM FOR TECHNOLOGICAL PROCESSES

Graduation Project
COM-400

Student: Aneel Ahsan KHAN (980662)

Supervisor: Asst. Prof. Dr. Rahib ABIYEV

Lefkoşa-2001

ACKNOWLEDGEMENT

First I want to thank Asst.Prof. Dr. Rahib ABIYEV to be my supervisor. Under his guidance, I successfully overcome many difficulties and learn a lot about Use of Neural Networks in Control Process. In each discussion, he explained my questions patiently, and I felt my quick progress from his advises.

I also want to thank my friends: Mustanser Siddque, Mudaser Siddque, Aun Shakeel and Rizwan Anjum.

Finally, I want to thank my family, especially my parents. Without their endless support and love for me. I would never achieve my current position and I wish my mother and my father live happily always.

ABSTRACT

The efficiency of technological processes particularly depends on the efficiency of control system used. The construction of control system on the base of traditional technology for complicated processes characterizing with non-linearity and uncertainty is not enough satisfy such characteristics as high speed, reliability, accuracy, adequacy of the model. In this condition one of perspective way of construction of control system is the use of neural technology that satisfy above characteristics of the system. The thesis is devoted to development of neural control system for technological processes. To solve the given problem at first stage the states of application problems of neural control system for technological processes are given, the mile stone achievements to the problem are clarified.

Using proposed structure the development of neural control system is performed. Controller is constructed on the base of neural network. The main problem of neural system synthesis is its learning. As a learning algorithm the "error back propagation algorithm" is chosen and its description is given.

Using learning algorithm and desired time response characteristics of the system the synthesis of neural controller for technological processes control is carried out. The modeling of the given system is performed. Results of simulations of the developed and traditional control system show the improved time response characteristics of previous.

TABLE OF CONTENTS

INTRODUCTION

By increasing complexity of the technological processes, uncertainty of an environment where technological processes take place the model of control system becomes very complicated. 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 are neural networks. Application of neural network for constructing control system allows us to increase their computation speed, validity, self-training and adapting capability.

Number of functions and possibilities of living organisms are realized by some neural structures. On the base of such structures one can get good models. Neural networks have such characteristics as: vitality, parallelism of computations, learning and generalization abilities, analytic description of linear and non-linear problems etc. Due to these characteristics Neural network becomes great of importance for application in such areas such as artificial behavior, artificial intelligence, theory of control and decision making, identification, optimal control, robotics etc.

One of perspective way of application of neural networks is the construction of control system for technological processes.

It is clear that the efficiency of using of any approach is defined by the criterions of calculations speed, reliability, vitality, flexibility, accuracy etc. If we consider previous systems, such as control systems developed on the base of traditional methods, control systems based on knowledge base (such as expert systems) considerable have disadvantage by their speed, vitality, adaptability, flexibility etc.

Because of it is necessary to develop intellectual control system on the base of neural network satisfying above characteristics.

The thesis consists of introduction, 3 chapters and conclusion.

Chapter 1 describes the state of application problems of artificial intelligence methods to solve control problems and the mile stone achievements to the problem.

Chapter 2 describes the architecture of neural control systems for technological processes. The structure of neural system and description of the functions of its main blocks are given. The mathematical models of neuron, neural network structures and their operation principle. The different neural network structures are described. The main problem in neural network is its learning. In the thesis the descriptions of the different learning algorithm are given. To train neural control system the supervised

learning algorithm- real time back propagation for recurrent neural network is chosen and its description is given.

Chapter 3 describes the development of neural control system for technological process. The desired time response characteristic of system, neural control system's learning algorithm and characteristics of the technological processes are described. Using these the synthesis procedures and simulation of neural control system are performed.

Conclusion presents the important obtained results and contributions in the thesis.

1. APPLICATIONS OF NEURAL NETWORKS FOR CONTROL PROBLEM

1.1. Introduction to Neural Computing and Categories of Neural Network Applications to Guidance and Control

"Future computer generation imitates man". "Many small cells are stronger than one large cell". Such headlines are to be found in the media in connection with a new kind of information processing, the so-called "Artificial Neural Networks (ANN)". As the term suggests, these networks are an attempt to-imitate the biological paradigm, our brain, in structure and function.

In the course of evolution our central nervous system (brain and spinal cord) has developed into a gigantic information-processing network to which the sensory paths from sense organs lead and from which the motor paths lead to the muscles. All stimuli are supplied to the central nervous system where they are processed into perceptions, sensations etc. and trigger off our actions.

In our organism many organ systems work together. Only the central nervous system communicates as superior system with all others-by collecting their information and coordinating their functions.

Basically similar problems will be found in future technical equipment and systems. Based on the structure of the biological brain, the creation of artificial neural networks (abbreviated ANN) is aimed to technically realize capabilities and characteristics such as self organization, learning and associative memory. This is achieved by the particular structure of neural networks where a large number of simple processor elements (PE) are interconnected with uni-directional signal channels to single or multi-layer networks. All processing elements are working in parallel as compared to one central, extremely efficient computer for sequential arithmetic and/or symbolic Information processing.

For the solution of a problem with a conventional computer (e.g. personal computer (PC)) an algorithm, a procedure-or a set of rules has to be developed and coded in software, i.e. a sequence of Instructions. These instructions are then carried out sequentially by the computer.

By contrast, ANNs are not programmed but trained and learn like their biological paradigm, the brain. This is done by changing the intensity of the connections between the

processor elements and by generating or eliminating structural connections. Thus the "knowledge" of an ANN lies in the topology and in the intensity of its connections, i.e. the strength of the connection weights between the PEs.

With their capabilities of self-organization, learning (adaptation) and association, ANNs can be used wherever it is difficult to describe a problem algorithmically, the development of the operational software is very cost-intensive or wherever unprecise, incomplete or even contradictory input data must be considered. Owing to the parallel information processing ANN are fault-tolerant and thus very reliable.

Ever-increasing requirements placed on more demanding and complex systems on the one hand and financial resources getting increasingly scarce on the other force us to tilter out key technologies showing the potential for a high cost-benefit ratio to meet the increased requirements. In this respect Artificial Neural Networks represent a new technology in the field of signal and information processing for Guidance and Control systems.

1.2. General Structure of Guidance and Control Problems

G.a.C. problems extend over several hierarchically structured levels and the communication functions between these levels as shown in Fig. 1.1.

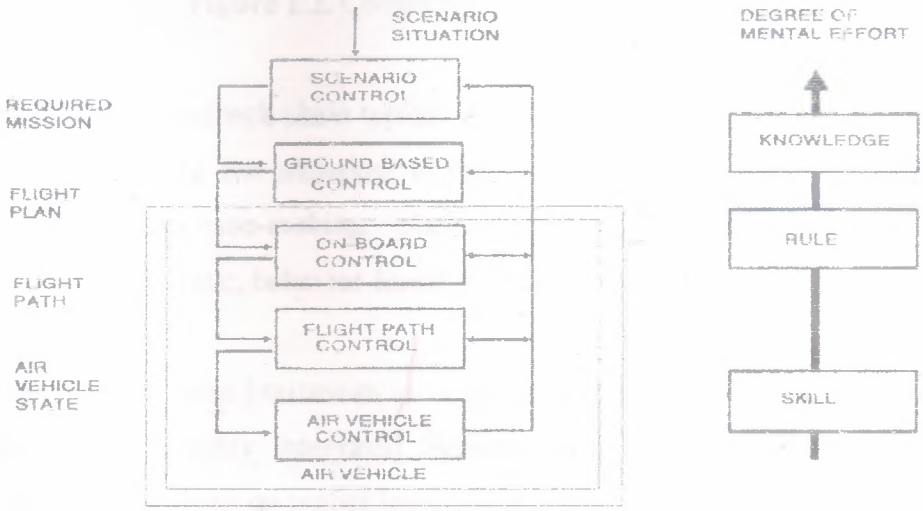

Figure 1.1. Cascaded G.a.C. Levels

The represented interconnection of the different function levels (scenario, mission, trajectory, air vehicle state) can be conceived of as a hierarchically structured control system. The objects on which G.a.C. functions are performed on the mentioned levels represent the control plants. Information processing by which actuation is generated on all levels from sensor information represents the controller which is of primary concern here (Fig. 1.2).

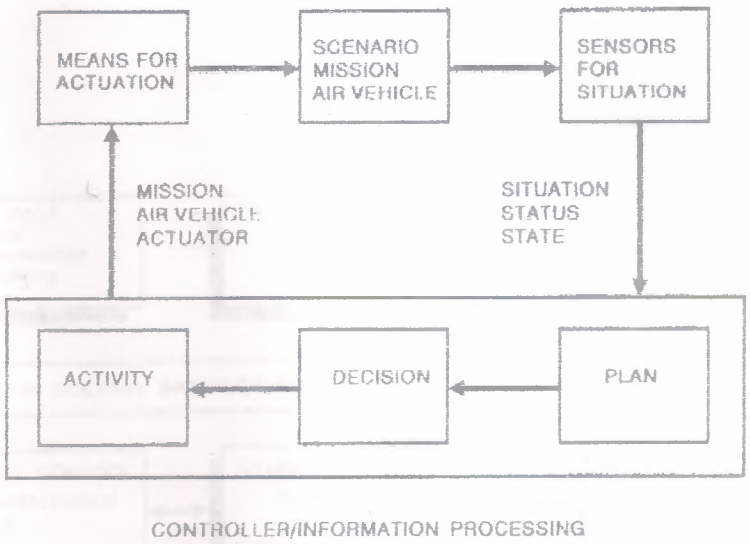

Figure 1.2. Common Structure of G.a.C. Levels

The controlling feedback chain typical of all G.a.C. levels requires functions such as recognizing and assessing the situation; defining action goals; generating optimum or favorable solutions; decision-making; planning and finally performing as well as monitoring of actions. Hence, behavior levels of mental capabilities can be assigned to the function levels (Fig. 1.1).

For reasons of human limitations in more demanding dynamic scenarios and in the operation of complex, highly Integrated Systems, there is the necessity for extended automation of these functions on higher levels such as trajectory control as well as mission management and control. Furthermore, the implementation of intelligent functions on lower levels such as the fusion and interpretation of sensor data, multifunctional use of sensor

information and smart/brilliant sensors become inevitable.

The technical implementation of the intelligent G.a.C. feedback chain functions leads to a signal processing structure which contains conventional arithmetic, symbolic and sub-symbolic elements (Fig.1.3). Whereas the symbolic element can be implemented utilizing expert-system software-techniques, the sub-symbolic element represents the application of ANNs. In building ANNs the brain is utilized as biological paradigm.

The following its function and structure are to be briefly explained as far as this is important for understanding ANNs.

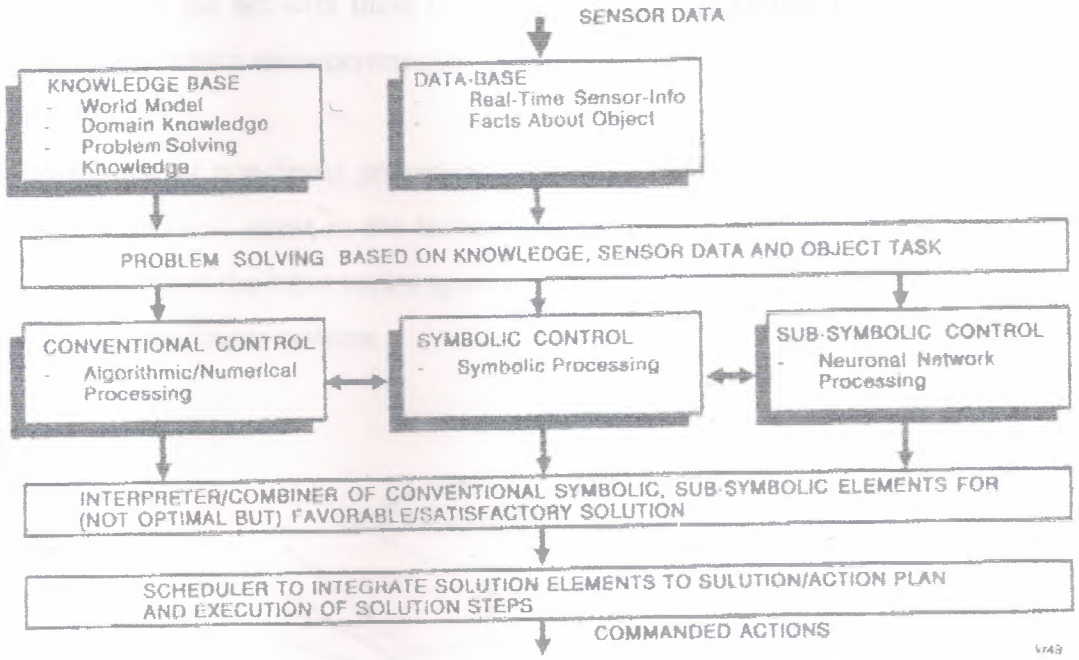

Figure 1.3. Activity chain implementation elements

1.3. General remarks for ANN application

Concerning the potential application it can be said that in many problems with only small or almost no knowledge existing on the object concerned, or where the parameters and states of this object can neither be described mathematically nor by rules and facts in a some how reliable manner, the development of sequential algorithms for conventional

processors is extremely difficult. The necessary expenditure of cost and time-for the algorithm and software development, verification and validation is correspondingly high.

Contrary to the sequential conventional information processing, the processing of information utilizing neural nets offers In general considerable advantages for all applications which are characterized by limited knowledge on the object. In contrast with the programmed sequential computing, ANN can be applied successfully for the solution of problems with inexact and incomplete or even contradictory input data.

The ability of neural nets to learn by examples (training patterns) or even unsupervised is of particular importance. It is not necessary to program a task-specific function or information. If representative example data are available in sufficient number and by training of the net with these data, due to its generalization property the net can tolerate input data which are superimposed by noise and disturbances, for the recognition of the input patterns.

By the use of non-linear processing elements in the network, multi-level nets can form complex decision -areas in the feature space. This corresponds mathematically to a non-linear mapping of the input vector space onto that of the output vector. This allows also the modeling of non-linear systems.

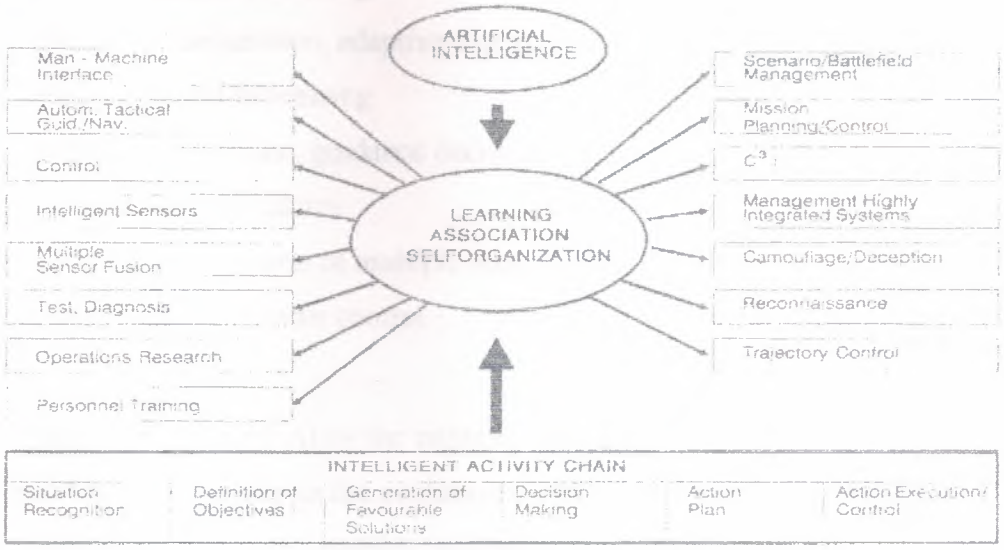

Figure 1.4. G.a.C. applications of Neural Computing and AI

1.4. Categories for ANN application

It has already been mentioned that G.a.C. problems extend over several hierarchically structured levels. In order to perform G.a.C. functions on these levels, the implementation of a controlling feedback chain typical of all G.a.C. levels is required. Technical apparatus which implements the feedback chain in a real-time autonomous system requires the solution of perception-problems as associated with the sensors and cognition problems (e.g. recognition, hypothesis testing) as far as the remaining functions are concerned. Very often in such systems, exploratory, goal-oriented actions will be performed resulting in a perception-cognition-action-recognition cycle.

It has been mentioned that for the implementation of such quasi mental functions elements of artificial intelligence are required. In addition to more conventional expert system techniques ANN will-gain an increasing importance within this scope. Therefore, the application potential for ANN covers many areas, extending from relatively simple applications in intelligent sensory and actuator systems to highly complex mission and scenario management problems.

Areas which represent potential categories for successful ANN application and which are recurring in many G.a.C. systems are the following:

- pattern recognition, signal classification associative memories
- self-organization, learning
- knowledge acquisition, adaptive-expert systems
- adaptive signal-processing
- control, stabilization, guidance decision finding
- optimization procedures
- integration and fusion of multiple sensor data
- robotics, sensory-motor control

1.5. Neuro Control

The application of ANN for control, stabilization and guidance of objects can be considered as a further step in the evolution control techniques to face up to the challenges within the scope of more complex systems which require more adaptation and self-organization capabilities. Thereby, the main problem Is concerned with the real-time

control of objects which are nonlinear and noisy and where the dynamics of which is time-varying, only incomplete or even unknown at all.

As common to all ANN, a characteristic feature of the neuro controllers is that they are not programmed but trained either supervised off-line or unsupervised on-line.

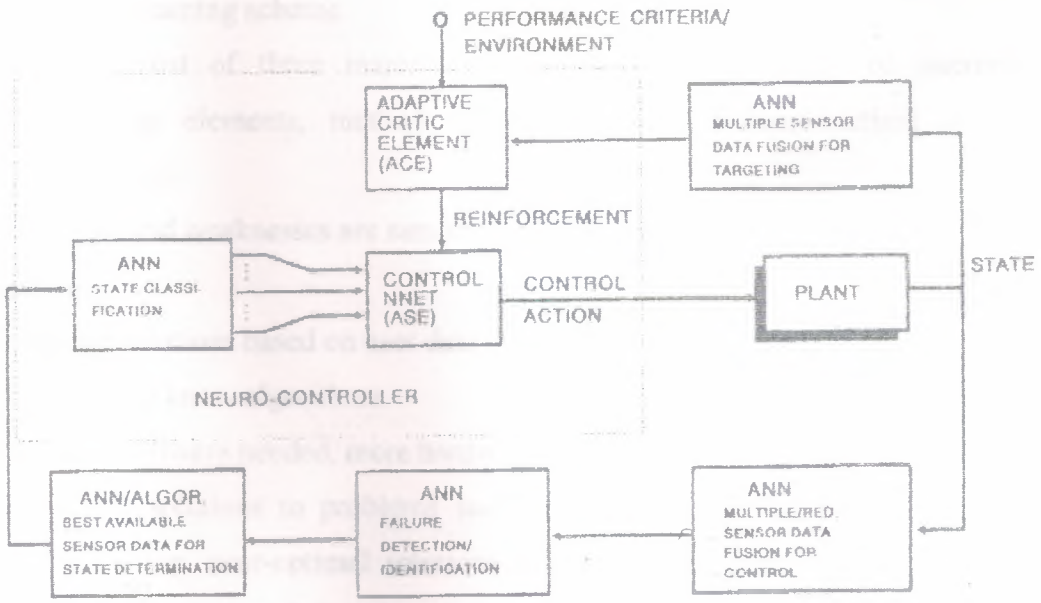

Figure 1.5. Neuro Control

As a generalized example the structure of a fault-tolerant, adaptive/learning neuro control system especially suitable for applications on the lower levels of the G.a.C. systems hierarchy (missiles, manned/unmanned air vehicles, robotics, mobile robots etc.) is shown in Fig. 1.5. As can be seen by this example, neuro-control systems can include subsystems for pattern recognition in sensor data, failure detection and identification, dynamic modeling etc. which are realized as ANN, however, are only of secondary importance for the actual neuro-control problem.

1.6. ANN Summary

The main features are as follows: Artificial neural nets

- are computers that learn how to solve problems.

- problem solving is based on sample data and learning mechanisms.
- they do not require expert knowledge representation, logical inferencing schemes, statistical algorithms specialist/analyst to develop and code a solution.
- they are trained to identify self-containedly the key features and associations enabling them to distinguish different patterns.
- can learn on-line real-time or can be trained off-line by a sample data set.
- do require an appropriate architecture with sufficient capacity and paradigmatic learning/training scheme.
- they consist of three major elements: organized topology of interconnected processing elements, method of encoding information, method of recalling information.

Their strengths and weaknesses are summarized as follows:

Strength:

- unique solutions based on user data examples
- no need to know algorithms
- less/no software needed, more hardware-processing power required
- provides solutions to problems such as: pattern matching and recognition, data compression, near-optimal solutions to optimization problems, non-linear system modeling and control, function approximation etc.
- inherent parallel processing structure yields faster solutions to a number of computation-intensive problems
- internal generation of complex decision areas by means of non-linear combination of input vector components robust performance in view of noisy and disturbed input signals
- inherently fault-tolerant

ANN weaknesses are that they are not applicable to all processing problems and do require training and test data examples -with' a few exceptions.

A comparison of ANN with conventional digital computers is summarized in table 1.1 This leads directly to some remarks regarding the utilization of ANNs.

Table 1.1: Comparison CONV. Digital Vs Neural processing

Feature	Digital Computer	Neural processing
Processing Order	Program with serially performed instructions	Parallel programs with comparatively few steps
Knowledge Storage	Static copy of knowledge is stored in addressed memory location. New information destroys old information	Information stored in the interconnections of the neurons Knowledge adapted by changing interconnection strength
Processing control	Central processing unit monitors all activities and has access to global information, creating processing bottleneck and critical point of failure	No control nor monitoring of a neuron activity Neuron's output only a function of its locally available information from interconnected neighbors
Fault Tolerance	Removal of any processing component leads to a defect Corruption of memory is irretrival, leads to a failure	Distributed knowledge/information representation across many neurons and their interconnection If portion of neurons removed, information retained through redundant distributed encoding
	FAULT INTOLERENT →	FAULT TOLERENT →

1.7. Applications of Artificial Neural Networks for Control Problem

In the design of autonomous computer-based systems, we often face the embarrassing situation of having to specify, to the system, how it should carry out certain tasks, which involve computations known to be intractable or are suspected of being so. To circumvent such impasses, we resort to complexity reducing strategies and tactics, which trade some loss of accuracy for significant reductions in complexity. The term

computational intelligence refers to such complexity reduction methods and to the research aimed at identifying such methods. In this paper we describe briefly some of our own work in this area and then develop a computational intelligence view of the task of process monitoring and optimization, as performed by autonomous systems. Some important current fields of discovery in computational intelligence include neural-net computing, evolutionary programming, fuzzy sets, associative memory and so on. [1]

Some of the theory-bound evolutional trends in real-time AI applications are pointed out based on analysis of essential properties of real-time Systems as well as AI based Systems [2]. The evolutional mainstream is increasing interdisciplinary integration. Three subtrends are illustrated on examples: mechanical combination of methods, A' methods used for approximate solution of classical problems, and abstract methods applied in new domains. In addition similarity between integrated circuits and real-time systems design and increased use of formal verification at the early stages of systems development are pointed out.

A new control system for the intelligent force control of multifingered robot grips which combines both fuzzy based adaptation level and neural based one with a conventional PID-controller [3]. The most attention is given to the neural based force adaptation level implemented by three layered back propagation neural networks. A computer based simulation system for the peg-in-hole insertion task is developed to analyze the capabilities of the neural controllers. Their behavior is discussed by comparing them to conventional and fuzzy based force controllers performing the same task.

Increasingly artificial neural networks are finding applications in process engineering environment. Recently the department of Trade and Industry in the UK has supported the transfer of neural technology to industry with a £5.7M campaign. As part of

the campaign, the University of the New Castle and EDS Advanced Technologies Group have setup a Process Monitoring and Control Club.[4]

This paper presents two case studies from the work of the Club. Firstly, the ability of neural networks to provide enhanced modeling performance over traditional linear techniques is demonstrated on real process data. Secondly, the ability of neural networks to capture non-linear system characteristics is exploited in a novel way in a condition monitoring exercise. The process studied in both applications is the melter stage of the BNFL Vitrification Process. The process involves the encapsulation of highly active liquid waste in glass blocks to provide a safe and convenient method of storage.

Process with (partly) unknown or complex dynamic behaviors need complex control schemes. Neural networks offer interesting perspectives both for identifications and control of these processes, because neural networks can approximate any (non-linear) continuous function. Especially adaptive control using neural networks offer good possibilities, due to the possibility to learn online. The NECTAR-project (Neural and Expert ConTrol of AircRaft) aims at studying both neural and expert systems structures for highly demanding control environments. [5]

A pilot study is described on the practical application of artificial neural networks. The limit cycle of the attitude control of a satellite is selected as the test case. One of the sources of the limit cycle is a position dependent error in the observed attitude. A Reinforcement Learning method is selected, which stable to adapt a controller such that a cost function is optimized. An estimate of the cost function is learned by a neural 'critic'. The estimated cost Unction is directly represented as a function of the parameters of a linear controller. The critic is implemented as a CMAC network. Results from simulations show that the method is able to find optimal parameters without unstable behavior. in

particular in the case of large discontinuities in the attitude measurements, the method shows a clear improvement compared to the conventional approach: the RMS attitude error decreases approximately 30%.[6]

The logical design of a neural controller is achieved by representing a neural computation as a stochastic timed linear proof with a built-in system for rewards and punishments based on the timeliness of a computation performed by a neural controller. Logical designs are represented with stochastic forms of proofnets and proofboxes. Sample applications of the logical design methodology to the truck-backer upper and a Real Time object recognition and tracking system (RTorts) are presented. Performance results of the implementation of the target dynamics identification module of the RTorts are given and compared to similar systems.[7]

2. STRUCTURE AND LEARNING OF NEURAL NETWORKS

2.1. Biological brain as paradigm, Artificial Neuron

Two different functions of the brain are to be looked at. First, there is the rational thinking with a function in conscious steps performed in a particular serial sequence. The digital computers we use today with a sequential processing of instructions listed in programs (computers in so-called von-Neumann architecture) were developed in the 1940s based on the investigation of sequentially conscious thinking.

On the other hand, there are the much more complex structures of unconscious thinking or unconscious intelligence. Here, a lot of environment data are processed within the context of our sensory perception and characteristics extracted. The sensorimotor control of our motions as well as three-dimensional thinking are largely unconscious. The structures of unconscious thinking provide the basis for the enormous capacity of our memory. All of these functions performed unconsciously are running parallel in networks in which so-called neurons interact due to a close interconnection and by means of electrochemical processes.

Our brain is organized as highly Integrated system in functional units, which are interconnected via variable connections, with each functional unit having about one thousand to one hundred thousand nerve cells. These each have ten to ten thousand equally variable, so-called synaptic connections to other neurons. In total, our central nervous system roughly contains the astronomical number of one hundred to one thousand billion nerve cells. It is clear that this enormous information-processing system cannot be completely structured and programmed prenatally even if genetic information is taken into account. The brain has the capability to organize itself, learn and establish associations.

To imitate biological Information processing models for different levels of organization and of abstraction have to be considered. First, there is the level of the individual neuron where it is a matter of representing the static and dynamic electrical characteristics as well as the adaptive behavior of the neuron. on the network level the Interconnection of identical neurons to form networks is examined to describe specific sensor and motoricity-related functions such as filtering, projection operations, controller function. In nonlinear, biological system, Networks on the mental function level are the

most complicated ones and comprise functions such as perception, solution of problems, strategic proceeding etc. These are the networks on the highest level of biological information processing.

2.1.1. The Biological Neuron

The most basic element of the human brain is a specific type of cell, which provides us with the abilities to remember, think, and apply previous experiences to our every action. These cells are known as neurons, each of these neurons can connect with up to 200000 other neurons. The power of the brain comes from the numbers of these basic components and the multiple connections between them. All natural neurons have four basic components, which are dendrites, soma, axon, and synapses. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then output the final result. The figure below shows a simplified biological neuron and the relationship of its four components.

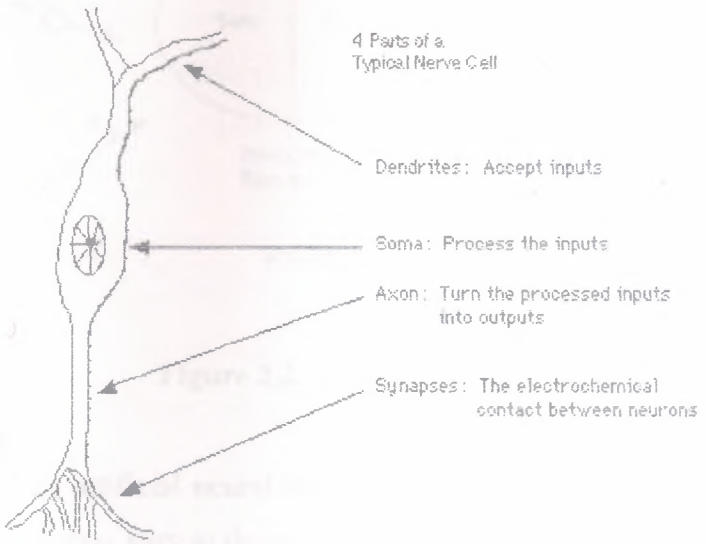

Figure 2.1. Biological Neuron

2.1.2. The Artificial Neuron

The basic unit of neural networks, the artificial neurons, simulates the four basic functions of natural neurons. Artificial neurons are much simpler than the biological neuron; the figure below shows the basics of an artificial neuron.

Note that various inputs to the network are represented by the mathematical symbol, x(n). Each of these inputs are multiplied by a connection weight, these weights are represented by w(n). In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output.

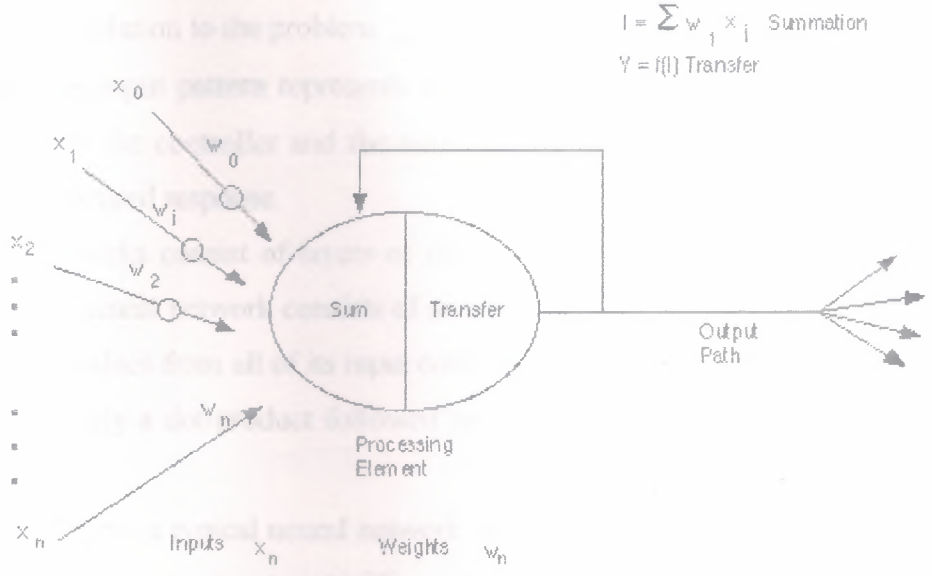

Figure 2.2. Artificial Neuron

Even though all artificial neural networks are constructed from this basic building block the fundamentals may vary in these building blocks and there are differences.

2.2. Neural Networks Characteristics

Neural networks are information processing systems. In general, neural networks can be thought of as "black box" devices that accept inputs and produce outputs. Some of the opera-dons that neural networks perform include:

- classification - an input pattern is passed to the network and the network produces a representative class as output.
- pattern matching - an input pattern is passed to the network and the network produces the corresponding output pattern.
- pattern completion - an incomplete pattern is passed to the network and the network produces an output pattern that has the missing portions of the input pattern filled in.
- noise removal - a noise-corrupted input pattern is presented to the network and the network removes some (or all) of the noise and produces a cleaner version of the input pattern as output.
- optimization - an input pattern representing the initial values for a specific optimization problem are presented to the network and the network produces a set of variables that represent a solution to the problem.
- control - an input pattern represents the current state of a controller and the desired response for the controller and the output is the proper command sequence that will create the desired response.

Neural networks consist of layers of processing elements and weighted connections. Each layer in a neural network consists of a collection of processing elements (PEs). Each PE collects the values from all of its input connections, performs a predefined mathematical operation (typically a dot-product followed by a threshold), and produces a single output value.

Figure 2.3 illustrates a typical neural network with three layers denoted F_x, F_y, and F_z. The bottom layer, F_x, accepts inputs into PEs x_1, x_2, x_3. A collection of weighted connections (sometimes called "weights" or "connections") connect the F_x PEs to the F_y PEs. The F_y PEs,y_1 and y_2, are the hidden layer. Similarly, the F_y PEs are connected to the Fz PEs which form the output layer. The weight names serve as both a label and a value.

As an example, in Figure 2.3 the connection from the Fx PE x_1 to the F_y PE y_2 is the connection weight w_{12} (the connection from x_1 to y_1. By adjusting the connection weights, information is stored in the network. The value of the connection weights are often determined by a neural network learning procedure (although sometimes they are predefined and hardwired into the network).By performing the update operations for each of the PEs the neural network recalls information.

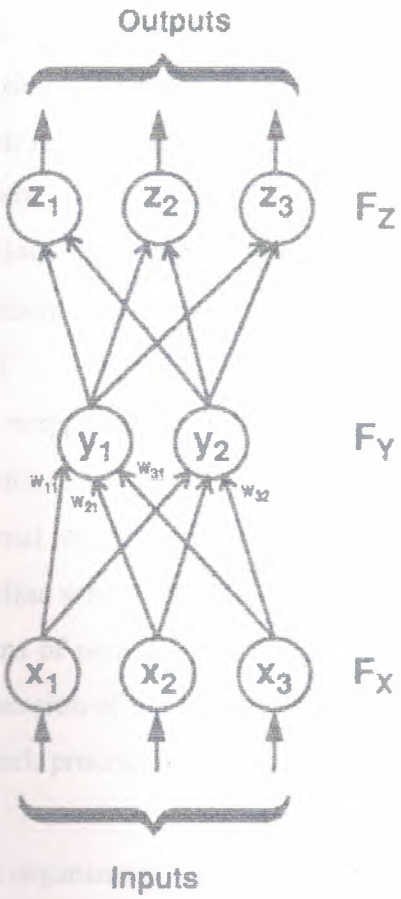

Figure 2.3. A typical Neuarl network

There are two important features illustrated by the neural network shown in Figure 2.3 that apply to all neural networks:

- Local Operations. Each PE acts independently of all others. A PE's output relies only on its constantly available inputs from the abutting connections. The information provided by the adjoining connections is all a PE needs to process. Information from other PEs where an explicit connection does not exist is not necessary.
- Distributed Representation. The large number of connections provides a large amount of redundancy and facilitates a distributed representation. A large number of connections must be eliminated for a significant amount of information to be destroyed.

The first feature allows neural networks to operate efficiently in parallel. The last

feature provides neural networks with inherent fault-tolerance and generalization qualities that are very difficult to attain from typical computing systems. In addition to these features, neural networks can learn arbitrary nonlinear mappings given the proper topology, nonlinear processing elements from nonlinear threshold operations, and appropriate learning rules. The ability to learn nonlinear mappings simply by presenting instances of input and output patterns is a powerful attribute shared by few systems.

There are three primary situations where neural networks are useful:

- Situations where only a few decisions are required from a massive amount of data(e.g. speech and image processing).
- Situations where nonlinear mappings must be automatically acquired (e.g. loan evaluations and robotic control).
- Situations where a near-optimal solution to a combinatorial optimization problem is required very quickly (e.g. airline scheduling and telecommunication message routing).

To summarize, the foundations of neural networks consist of an understanding of the nomenclature and a firm comprehension of the rudimentary mathematical concepts used to describe and analyze neural network processing. In a broad sense, neural networks consist of three principle elements:

- Topology. A neural network's organization into interconnected layers.
- Learning. The adjustment of weights to store information.
- Recall. Retrieving information stored in the weights.

2.3. Dissecting Neural Networks

A convenient neural network analogy is the directed graph, where the edges and nodes correspond to weights and PEs, respectively. In addition to connections and processing elements, threshold functions and input/output patterns are also basic elements in the design, implementation and use of neural networks. After a description of the terminology used to describe neural networks, each of these elements will be examined in turn.

2.3.1. Terminology

Unfortunately, neural network terminology remains varied, with a standard yet to be adopted. To illustrate some of the terminology introduced here, please refer to Figure 2.4

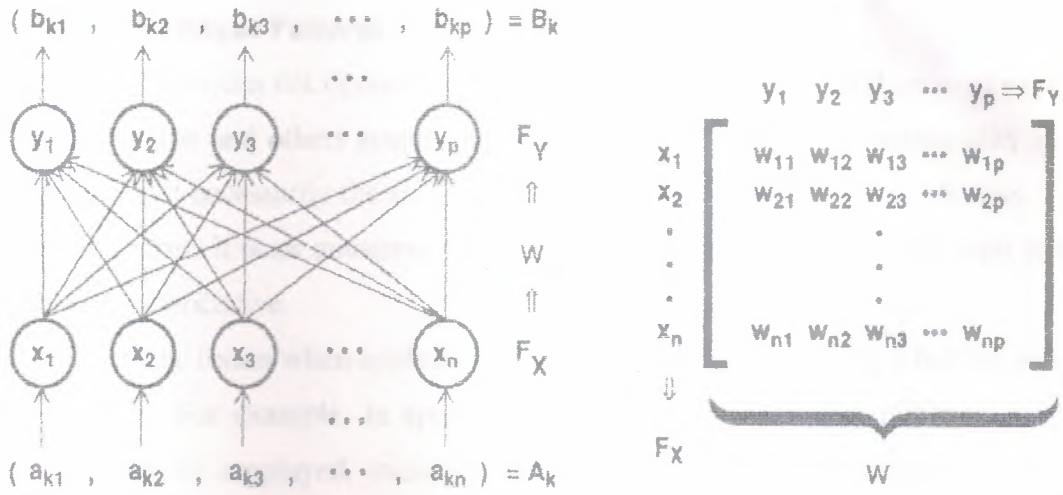

Figure 2.4. Two Layer Feed Forward Network and Weight Matrix

Input and output vectors (patterns) are denoted by subscripted capital letters from the beginning of the alphabet. The input m patterns are denoted as $A_k = (a_{k1}, a_{k2}, ..., a_{kn})$; k = 1, 2,3....m., and the output patterns as $B_k = (b_{k1}, b_{k2}.... b_{km})$; k = 1,2,3....m.

The PEs in a layer will be denoted by the same subscripted variable. The collection of PEs in a layer form a vector and these vectors will be denoted by capital letters from the end of the alphabet. In most cases three layers of PEs will suffice. The input layer of PEs is denoted as $F_x = (x_1, x_2, ..., x_n)$, where each x_i receives input from the corresponding input pattern component a_{ki}. The next layer of PEs will be the F_y PEs, then the F_z PEs (if either layer is necessary). The dimensionality of these layers depends on its use. Using the network in Figure 2.4 as an example, the second layer of the network is the output layer, hence the number of F_y PEs must match the dimensionality of output patterns. In this instance, the output layer is denoted as $F_y = (y_1, y_2, ...,y_p)$, where each y_j is correlated with the j'th element of B_k.

Connection weights are stored in weight matrices. Weight matrices will be denoted by capital letters toward the middle of the alphabet, such as U, V, and W. Referring to the example in Figure 2.4 this two layer neural network requires one weight matrix to fully connect the layer of n F_x PEs to the layer of p F_y PEs. The matrix shown in Figure 2.4

describes the full set of connection weights between F_x and F_y, where the weight w_{ij} is the connection weight from the i'th F_x PE, x_i, to the j'th F_y PE, y_j

2.3.2. Input and Output Patterns

Neural networks can not operate unless they have data. Some neural networks require only single patterns and others require pattern pairs. Note that the dimensionality of the input pattern is not necessarily the same as the output pattern. When a network only works with single patterns, it is an autoassociative network. When a network works with pattern pairs it is heteroassociative.

One of the key issues when applying neural networks is determining what the patterns should represent. For example, in speech recognition there are many different types of features that can be employed, including: linear predictive coding coefficients, Fourier spectra, histograms of threshold crossings, cross-correlation values. The proper selection and representation of these features can greatly affect the performance of the network.

2.3.3. Connections

A neural network is equivalent to a directed graph (digraph). A digraph has edges (connections) between nodes (PEs) that allow information to flow in only one direction (the direction denoted by the arrow). Information flows through the digraph along the edges and is collected at the nodes. Within the digraph representation, connections determine the direction of information flow. As an example, in Figure 2.2 the information flows from the F_X layer through the connections, W, to the F_y layer. Neural networks extend the digraph representation to include a weight with each edge (connection) that modulates the amount of output signal passed from one node (PE) down the connection to the adjacent node. For simplicity, the dual role of connections will be employed. A connection both defines the information flow through the network and it modulates the amount of information passing between to PEs.

The connection weights are adjusted during a learning process that captures information. Connection weights that are positive valued are excitatory connections. Those that with negative values are inhibitory connections. A connection weight that has a zero value is the same as not having a connection present. By only allowing a subset of all the possible connections to have non-zero values, sparse connectivity between PEs can be simulated.

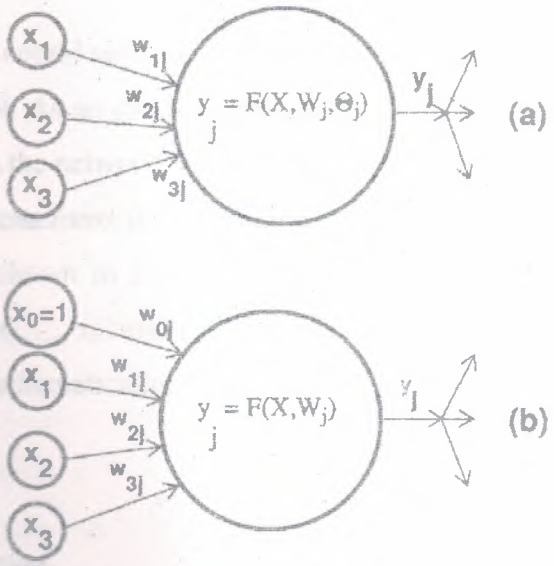

Figure 2.5. The Processing Element

It is often desirable for a PE to have an internal bias value (threshold value). Panel (a) of Figure 2.5 shows the PE y_j with three connections from F_X (w_1, w_2~ w_3) and a bias value, θj . It is convenient to consider this bias value as an extra connection, w_{Oj}, emanating from the F_x PE x_0, with the added constraint that x~ is always equal to 1 as shown in panel (b). This mathematically equivalent representation simplifies many discussions.

2.3.4. Processing Elements

The processing element (PE) is the portion of the neural network where all the computing is performed. Figure 2.5 illustrates the most common type of PE. A PE can have one input connection, as is the case when the PE is an input layer PE and it receives only one value from the corresponding component of the input pattern, or it can have several weighted connections, as is the case of the F_y PEs shown in Figure 2.5 where there is a connection from every F_x PE to each F_y PE. Each PE collects the information that has been sent down its abutting connections and produces a single output value. There are two important qualities that a PE must possess:

Local Operations: Described earlier

Single Output Value: Each PE produces a single output value that is propagated through the connections from the emitting PE to other receiving PEs or it will be output from the network.

These two qualities allow neural networks to operate in parallel. The value of the PE and its label use the same symbol. As an example, the output PE label y_j in Figure 2.5 represents both the PEs placement in the network and its value.

There are several mechanisms for computing the output of a processing element. The output value of the PE shown in Figure 2.5(b), y_j is a function of the outputs of the preceding layer, $F_x = (x_1, x_2, ..., x_n)$ and the weights from F_X to y_j

$Wj=(w_{1j}, w_{2j}, ..., w_{nj})$. Mathematically, the output of y_j is a function of its inputs and its weights,

$$y_j = F(X, W_j) \tag{2.1}$$

2.3.4.1. Linear Combination

The most common computation performed by a PE is a linear combination (dot-product) of the input values, X, with the abutting connection weights, W_j followed by a threshold operation. Using the PE in Figure 2.5(b) as an example, the output y_j is computed using the equation.

$$y_j = f\left(\sum_{i=0}^{n} x_i w_{ij}\right) = f(X \bullet W_j) \tag{2.2}$$

where $W_j = (W_{1j}, W_{2j}.....W_{nj})$ and f is one of the threshold functions. The dot product update has a very appealing quality that is intrinsic to its computation. Using the relationship $A_k \bullet W_i = \cos(A_k.W_j)\|A_k\|\|W_j\|$ it is seen that the larger the dot product (assuming fixed length A_k and W_j) the more similar the two vectors are. Hence, the dot product can be viewed as a similarity measure.

2.3.4.2. Mean-Variance Connections

In some instances PEs will have two connections interconnecting PEs instead of just one as shown in Figure 2.6.

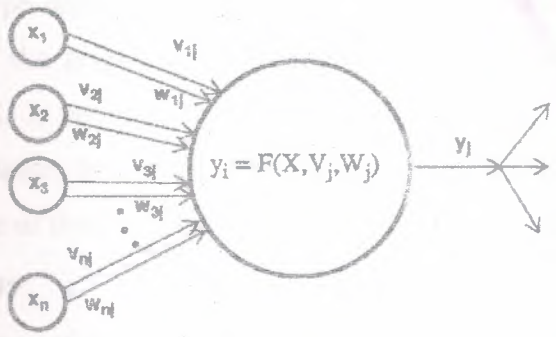

Figure 2.6. Dual Connections

One use of these dual connections is to allow one set of the abutting connections represent the mean of a class and the other the variance of the class. In this case, the output value of the PE depends on the inputs and both sets of connections, i.e. $y_j=F(X, V_j, W_j)$ where the mean connections are represented by $W_j = (w_{1j}, w_{2j}, ..., w_{nj})$ and the variance connections $V_j=(v_{1j},v_{2j}, ..., v_{nj})$ for the PE y_j

2.3.4.3. Min Max Connections

Another less common use of dual connections is to assign one of the abutting vectors, say V_j to become the minimum bound for the class and the other vector, W_j, to becomes the maximum bound for the same class. By measuring the amount of the input pattern that falls within the bounds, a min-max activation value is produced. Figure 2.7 illustrates this notion using a graph representation for the min and the max points.

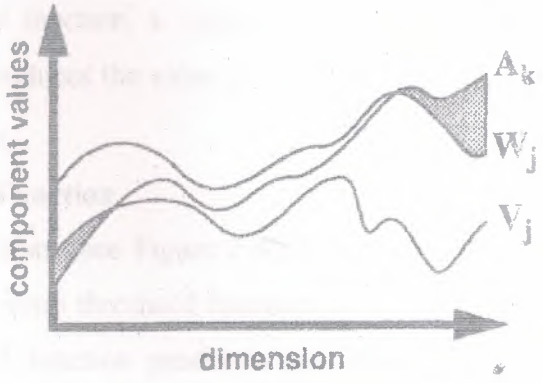

Figure 2.7. Min-Max Classification

The ordinate of the graph represents the value of each element of the min and max vectors and the abscissa of the graph represents the dimensionality of the classification space. The input pattern, X, is compared with the bounds of the class. The amount of disagreement between the classes bounds, V_j and W_j, and input pattern, X, is shown in the shaded regions. The measure of these shaded regions produces an activation value y_j.

2.4. Threshold functions

Threshold functions, also referred to as activation functions, squashing functions, or signal functions, map a PE's (possibly) infinite domain to a prespecified range. Although the number of threshold functions possible is quite varied, there are five that are regularly employed by the majority of neural~ networks:

(1) linear, (2) step, (3) ramp, (4) sigmoid, and (5) Gaussian. With the exception of the linear threshold function, all of these introduce a non-linearity in the network dynamics by bounding a PE's output values to a fixed range.

2.4.1. Linear Threshold Function

The linear threshold function (see Figure 2.8(a)), produces a linearly modulated output from the input x as described by the equation

$$f(x) = \propto x \qquad (2.3)$$

where x ranges over the real numbers and a is a positive scalar. if $\alpha = 1$, it is equivalent to removing the threshold function completely.

2.4.2. Step Threshold Function

The step threshold function, (see Figure 2.8(b)), produces only two values, β and δ .If the input to the threshold function, x, equals or exceeds the threshold value, θ, then the step threshold function produces the value β, otherwise it produces the value $-\delta$, where β and α are positive scalars.

2.4.3. Ramp Threshold Function

The ramp threshold function, (see Figure 2.8(c)), is a combination of the linear and step threshold functions. The ramp threshold function places an upper and lower bound on the values that the threshold function produces and allows a linear response between the bounds. These saturation points are symmetric around the origin and are discontinuous at the points of saturation.

2.4.4. Sigmoid Threshold Function

The sigmoid threshold function, (see Figure 2.8(d)), is a continuous version of the ramp threshold function. The sigmoid (S-shaped) function is a bounded, monotonic, non-decreasing function that provides a graded, nonlinear response within a prespecified range.

The most common sigmoid function is the logistic function

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \tag{2.4}$$

where ($\alpha > 0$ (usually $\alpha = 1$), which provides an output value from 0 to 1.

2.4.5. Gaussian Threshold Function

The Gaussian threshold function, (see Figure 2.8(e)), is a radial function (symmetric about the origin) that requires a variance value, v >0, to shape the Gaussian function. In some networks the Gaussian function is used in conjunction with a dual set of connections

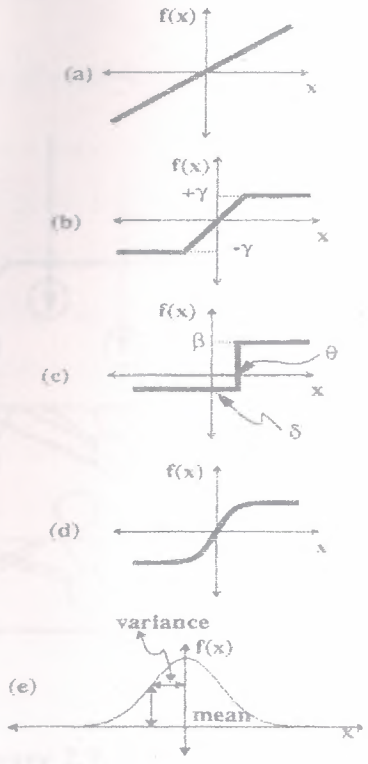

Figure 2.8. Threshold Functions

2.5. Neural Network Topologies

The building blocks for neural networks are in place. Neural networks consist of layer(s) of PEs interconnected by weighted connections. The arrangement of the PEs, connections and patterns into a neural network is referred to as a topology.

2.5.1. Layers

Biologically, neural networks are constructed in a three dimensional way from microscopic components. These neurons seem capable of nearly unrestricted interconnections. This is not true in any man-made network. Artificial neural networks are the simple clustering of the primitive artificial neurons. This clustering occurs by creating layers, which are then connected to one another. How these layers connect may also vary. Basically, all artificial neural networks have a similar structure of topology. Some of the neurons interface the real world to receive its inputs and other neurons provide the real world with the network's outputs. All the rest of the neurons are hidden form view.

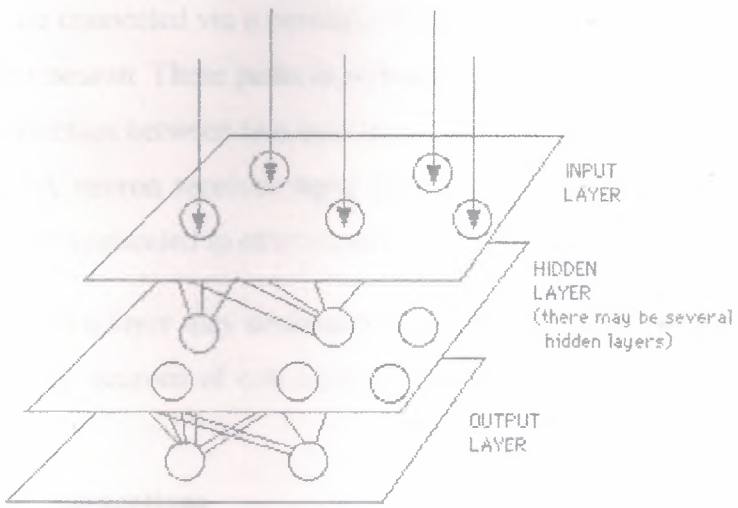

Figure 2.9. layers structure

As the figure above shows, the neurons are grouped into layers The input layer consist of neurons that receive input form the external environment. The output layer consists of neurons that communicate the output of the system to the user or external environment. There are usually a number of hidden layers between these two layers; the figure above shows a simple structure with only one hidden layer.

When the input layer receives the input its neurons produce output, which becomes input to the other layers of the system. The process continues until a certain condition is satisfied or until the output layer is invoked and fires their output to the external environment.

To determine the number of hidden neurons the network should have to perform its best, one are often left out to the method trial and error. If you increase the hidden number of neurons too much you will get an over fit, that is the net will have problem to generalize. The training set of data will be memorized, making the network useless on new data sets.

2.5.2 Communication and types of connections

Neurons are connected via a network of paths carrying the output of one neuron as input to another neuron. These paths is normally unidirectional, there might however be a two-way connection between two neurons, because there may be an another path in reverse direction. A neuron receives input from many neurons, but produce a single output, which is communicated to other neurons.

The neuron in a layer may communicate with each other, or they may not have any connections. The neurons of one layer are always connected to the neurons of at least another layer.

2.5.2.1 Inter-layer connections

There are different types of connections used between layers, these connections between layers are called inter-layer connections.

- **Fully connected** Each neuron on the first layer is connected to every neuron on the second layer.

- **Partially connected** A neuron of the first layer does not have to be connected to all neurons on the second layer.

- **Feed forward** The neurons on the first layer send their output to the neurons on the second layer, but they do not receive any input back form the neurons on the second layer.

- **Bi-directional** There is another set of connections carrying the output of the neurons of the second layer into the neurons of the first layer.

Feed forward and bi-directional connections could be fully or partially connected.

- **Hierarchical** If a neural network has a hierarchical structure, the neurons of a lower layer may only communicate with neurons on the next level of layer.

- **Resonance** The layers have bi-directional connections, and they can continue sending messages across the connections a number of times until a certain condition is achieved.

2.5.2.2 Intra-layer connections

In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of intra-layer connections.

- **Recurrent** The neurons within a layer are fully- or partially connected to one another. After these neurons receive input form another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer. Generally some conditions among the neurons of the layer should be achieved before they communicate their outputs to another layer.

- **On-center/off surround** A neuron within a layer has excitatory connections to itself and its immediate neighbors, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal interchange, the neurons with an active output value will win, and is

allowed to update its and its gang member's weights. (There are two types of connections between two neurons, excitatory or inhibitory. In the excitatory connection, the output of one neuron increases the action potential of the neuron to which it is connected. When the connection type between two neurons is inhibitory, then the output of the neuron sending a message would reduce the activity or action potential of the receiving neuron. One causes the summing mechanism of the next neuron to add while the other causes it to subtract. One excites while the other inhibits.)

2.6. Single-layer Networks: Autoassociation, Optimization, and Contrast Enhancement

Beyond the instarloutstar neural networks are the single layer intraconnected neural networks. Figure 2.10 shows the topology of a one-layer neural network which consists of n F_x PEs. The connections from each F_x PE to every other Fx PE and itself, yielding a connection matrix with n^2 entries. The single-layer neural network accepts an n-dimensional input pattern in one of three ways:

- PE Initialization Only. The input pattern is used to initialize the Fx PEs and the input pattern does not influence the processing thereafter.

PE Initialization and Constant Bias. The input pattern is use to initialize the F_x PEs and the input remains as a constant valued input bias throughout processing.

Constant Bias Only. The PEs are initialized to all zeroes and the input pattern acts as a constant valued bias throughout processing.

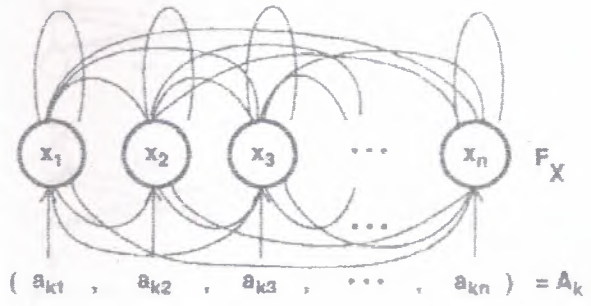

Figure 2.10. Single layer Neural Network

One-layer neural networks are used for pattern completion, noise removal, optimization, and contrast enhancement. The first two operations are performed by autoassociatively encoding patterns and typically using the input pattern for PE initialization only. The optimization networks are dynamical systems that stabilize to a state that represents a solution to an optimization problem and typically uses the inputs for both PE initialization and as constant biases. Contrast enhancement networks use the input patterns for PE initialization only and can operate in such a way that eventually only one PE remains active. Each of these one-layer neural networks are described in greater detail in the following paragraphs.

2.7.Multi-layer Networks: Heteroassociation and Function approximation

A multi-layer neural network has more than two layers, possibly many more. A general description of a multi-layer neural network is shown in Figure 2.11, where there is an input layer of PEs, Fx, L hidden layers of Fy PEs and a final output layer, F_Z. The F_y layers are called hidden layers because there are no direct connections between the input/output patterns to these PEs, rather they are always accessed through another set of PEs such as the input and output PEs. Although Figure shows connections only from one layer to the next, it is possible to have connections that skip over layers, that connect the input PEs to the output PEs, or that connect PEs together within the same layer. The added benefit of these PEs is not fully understood, but many applications such as prediction and classification are employing these types of topologies.

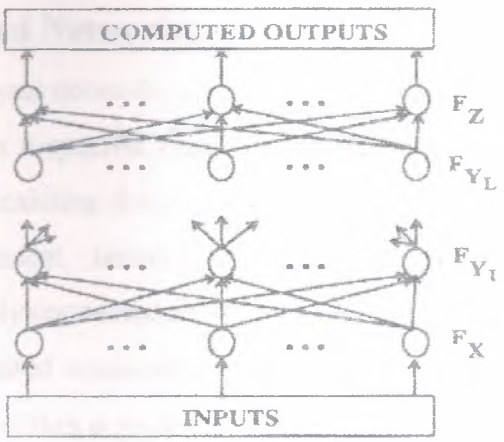

Figure 2.11. General Multi-layer Neural Network

Multi-layer neural networks are used for pattern classification, pattern matching and function approximation. By adding a continuously differentiable threshold function, such as a Gaussian or sigmoid function, it is possible to learn practically any nonlinear mapping to any desired degree of accuracy. The mechanism that allows such complex mappings to be acquired is not fully understood for each type of multi-layer neural network, but in general the network partitions the input space into regions and a mapping from the partitioned regions to the next space is performed by the next set of connections to the next layer of PEs, eventually producing an output response. This capability allows some very complex decision regions to be performed for classification and pattern matching problems, as well as applications that require function approximation.

There are several issues that must be addressed when working with multi-layer neural networks. How many layers is enough for a given problem? How many PEs are needed in each hidden layer? How much data is needed to produce a sufficient mapping from the input layer to the output layer? Some of these issues have been successfully dealt with. As an example, there have been several researchers that have proven that three layers is sufficient to perform any nonlinear mapping (with the exception of a few remote pathological cases) to any desired degree of accuracy with only one layer of hidden PEs. Although this is a very important result, it still does not indicate what the proper number of hidden layer PEs is, or if the same solution can be obtained with more layers but fewer hidden PEs and connections overall.

2.8. Randomly Connected Networks

Randomly connected neural networks are networks that have connection weights that are randomly assigned within a specific range. Some randomly connected networks have binary valued connections. Realizing that, a connection weight equal to zero is equivalent to no connection being present, binary valued random connections create sparsely connected networks. Randomly connected networks are used in three different ways:

- Initial weights The initial connection values for the network prior to training are pre-set to random values within a predefined range. This technique is used extensively in error-correction learning system.
- Pattern preprocessing - A set of fixed random binary valued connections are placed

between the first two layers of a multi-layer neural network as a pattern preprocessor. The use of such random connections can be used to increase the dimensionality of the space that is being used for mappings in an effort to improve the pattern mapping capability. This approach was pioneered with the early Perceptron and has been used recenfly in the Sparse Distributed Memory.

- Intelligence from randomness - Early studies in neural networks spent a great deal of effort analyzing randomly connected binary valued systems. The model of the brain as a randomly connected network of neurons prompted this research.

2.9. Neural Network Learning

Perhaps the most appealing quality of neural networks is their ability learn. Learning, in this context, is defined as a change in connection weight values that results in the capture of information that can later be recalled. There are several different procedures available for changing the values of connection weights. After an introduction to some terminology, eight different learning methods will be described. For continuity of discussion, the learning algorithms will be described in point-wise notation (as opposed to vector notation). In addition, the learning algorithms will be described using discrete time equations (as opposed to continuous time). The use of discrete-time equations makes them more accessible to digital computer simulations.

2.9.1. Supervised vs. Unsupervised Learning

All learning methods can be classified into two categories, supervised learning and unsupervised learning. Supervised learning is a process that incorporates an external teacher and/or global information. The supervised learning algorithms that will be discussed in the following sections include error correction learning, reinforcement learning, stochastic learning, and hardwired systems. Examples of supervised learning include; deciding when to turn off the learning, deciding how long and how often to present each association for training, and supplying performance (error) information. Supervised learning is further classified into two subcategories; structural learning and temporal learning. Structural learning is concerned with finding the best possible input/output relationship for each individual pattern pair. Examples of structural learning include pattern matching and pattern classification. The majority of the learning algorithms discussed below focus on structural learning. Temporal learning is concerned with capturing a

sequence of patterns necessary to achieve some final outcome. In temporal learning the current response of the network is dependant on previous inputs and responses. In structural learning, there is no such dependence. Examples of temporal learning include prediction and control. The reinforcement learning algorithm discussed below is an example of a temporal learning procedure.

Unsupervised learning, also referred to as self-organization, is a process that incorporates no external teacher and relies upon only local information during the entire learning process. Supervised learning organizes presented data and discovers its emergent collective properties. Examples of unsupervised learning that will be discussed in the following sections includes Hebbian learning, principle component learning, differential Hebbian learning, min-max learning, and competitive learning.

2.9.2. Off-line vs. On-line Learning

Most learning techniques utilize off-line learning. When the entire pattern set is used to condition the connections prior to the use of the network, it is called off-line learning. As an example, the backpropagation training algorithm is used to adjust connections in multi-layer neural network, but it requires thousands of cycles through all the pattern pairs until the desired performance of the network has been achieved. Once the network is performing adequately, the weights are frozen and the resulting network is used in recall mode thereafter. Off-line learning systems have the intrinsic requirement that all the patterns have to be resident for training. Such a requirement does not make it possible to have new patterns automatically incorporated into the network as they occur, rather these new patterns must be added to the entire set of patterns and a retraining of the neural network must be done again.

Not all neural networks perform off-line learning. There are some networks that can add new information "on the fly" non-destructively. If a new pattern needs to be incorporated into the network's connections, it can be done

2.9.3. Hebbian Correlations

The simplest form of adjusting connection weight values in a neural network is based upon the correlation of PE activation values. The motivation for correlation-based adjustments has been attributed to Hebb (1949) who hypothesized that the change in a synapses efficacy (its ability to fire, or as we are simulating it in our neural networks, the

connection weight) is prompted by a neuron's ability to produce an output signal. If a neuron, A, was active, and A's activity caused a connected neuron, B, to fire, then the efficacy of the synaptic connection between A and B should be increased.

2.9.4. Principle Component Learning

There are some neural networks that have learning algorithms designed to produce, as a set of weights, the principle components of the put data patterns. The principle components of a set of data are found by forming the covariance (or correlation) matrix of a set of patterns and then finding the minimal set of orthogonal vectors that span the space of the covanance matrix. Once the basis set has been found, it is possible to reconstruct any vector in the space with a linear combination of the basis vectors. The value of each scalar in the linear combination represents the "importance" of that basis vectors (Lawley & Maxwell, 1963). It is possible to think of the basis vectors as feature vectors and the combination of diese feature vectors is used to construct patterns. Hence, the purpose of a principle component network is to decompose an input pattern into values the represent the relative importance of the features underlying the patterns.

2.9.5. Differential Hebbian Learning

Hebbian learning has been extended to capture the temporal changes that occur in pattern sequences. This learning law, entitled Differential Hebbian Learning, has been independently derived by Klopf (1986) in the discrete time form and by Kosko (1986) in the continuous time form. The general form, some variants, and some similar learning laws are outlined in the following sections. There are several other combinations that have been explored beyond those that are presented in this section.

2.9.6. Competitive Learning

Competitive learning is a method of automatically creating classes for a set of input patterns. Competitive learning is a two step procedure that couples the recall process with the learning process in a two layer neural network (see Figure 2.12). In Figure 2.12 each F_x PE represents a component of the input pattern and each Fy PE represents a class

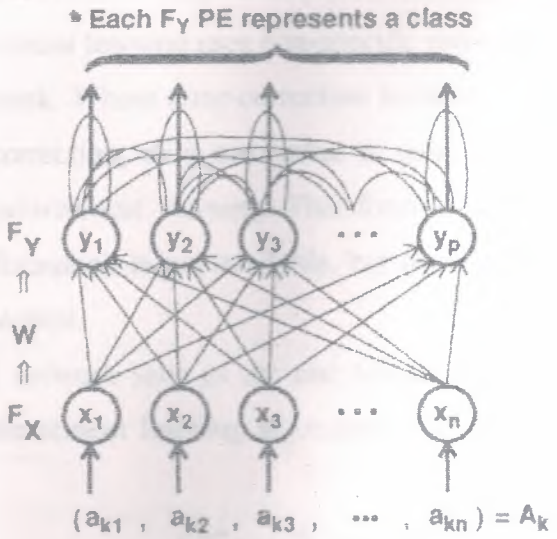

Figure 2.12. Competitive Learning neural network

2.9.7. Min-Max Learning

Min-max classifier systems utilize a pair of vectors for each class For the classj, represented by the PE y_j and defined by the abutting vectors V_j (the min vector) and (the max vector). Learning in a min-max neural system is done using the equation

$$v_{ij}^{new} = \min(a_k i, v_{ij}^{old}) \quad (2.5)$$

for the min vector and

$$v_{ij}^{new} = \max(a_k i, v_{ij}^{old}) \quad (2.6)$$

for the max vector.

2.9.8. Error Correction Learning

Error correction learning adjusts the connection weights between PEs in proportion to the difference between the desired and computed values of each output layer PE. Two layer error correction learning is able to capture linear mappings between input and output patterns. Multi-layer error correction learning is able to capture nonlinear mappings between the inputs and outputs.

2.9.9. Reinforcement Learning

Reinforcement learning is similar to error correction learning in that weights are reinforced for properly performed actions and punished for poorly performed actions. The

difference between these two supervised learning techniques is that error correction learning utilizes more specific error information by collecting error values from each output layer PE, while reinforcement learning uses non-specific error information to determine the performance of the network. Where error-correction learning has a whole vector of values that it uses for error correction, only one value is used to describe the output layer's performance during reinforcement learning. This form of learning is ideal in situations where specific error information is not available, but overall performance information is, such as prediction and control.

A two-layer neural network such as the one found in Figure 2.13 serves as a good framework for the reinforcement learning algorithm. The general reinforcement learning equation is

$$w_{ij}^{new} = w_{ij}^{old} + \alpha(r - \theta_j)e_{ij} \qquad (2.7)$$

where, r is the scalar success/failure value provided by the environment, θ_j is the reinforcement threshold value for the j'th F_y PE, e_{ij}; is the canonical eligibility of the weight from the i'th F_x PE to the j'th FYPE, and $O<\alpha<1$ is a constant-valued learning rate. In error correction learning, gradient descent in error space controlled learning. In reinforcement learning it is gradient descent in probability space.

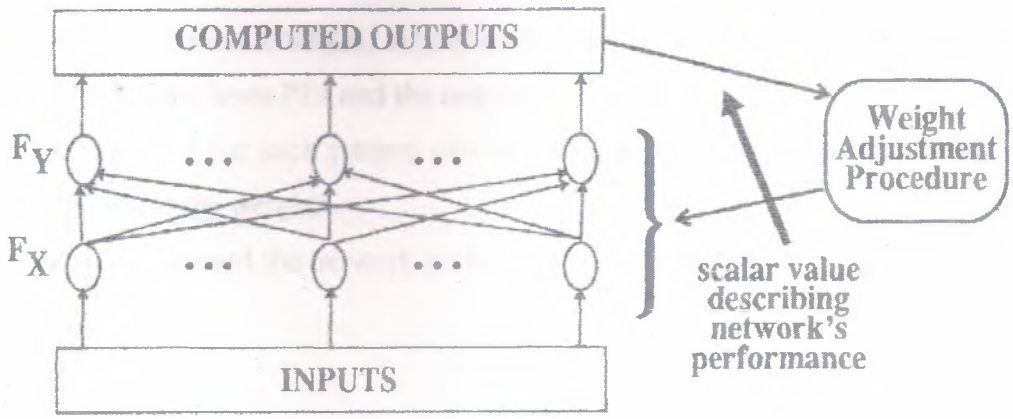

Figure 2.10. Reinforcement learning Neural Network

The canonical eligibility of w_{ij} is dependant on a previously selected probability distribution that is used to determine if the computed output value equals the desired output value and is defined as

$$e_{ij} = \frac{\partial}{\partial w_{ij}} \ln g_i \tag{2.8}$$

where g_j is the probability of the desired output equaling the computed output, defined as

$$g_i = \Pr\left(y_j = b_{kj} \middle| W_j, A_k\right) \tag{2.9}$$

which is read as the probability that yj equals b_{kj} given the input, A_k, and the corresponding weight vector, W_j.

2.9.10. Stochastic Learning

Stochastic learning uses random processes, probability, and an ene~y relationship to adjust connection weights in a multi-layered neural network. Using the three-layer neural network shown in Figure 2.14 to illustrate the learning algorithm, the stochastic learning procedure is described as follows:

1. Randomly change the output value of a hidden layer PE (the hidden layer PEs utilize a binary step threshold function).
2. Evaluate the change using the resulting difference in the neural network's energy as a guide. If the energy after the change is lower, keep the change. If the change in energy is not lower after the random change, accept the change according to a pre-chosen probability distribution.
3. After several random changes, the network will eventually become "stable." Collect the values of the hidden layer PEs and the output layer PEs.
4. Repeat steps 1-3 for each pattern pair in the data set, then use the collected values to statistically adjust the weights.
5. Repeat steps 1-4 until the network performance is adequate.

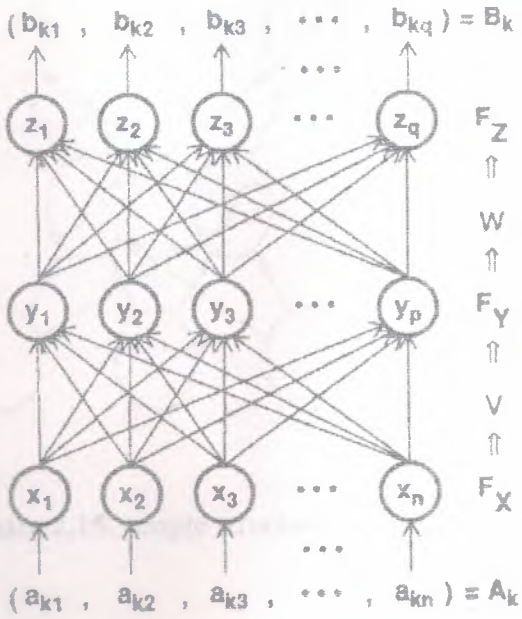

Figure 2.14. Three Layer Network

2.9.11. Error Backpropagation

We have already seen how to train linear networks by gradient descent. In trying to do the same for multi-layer networks we encounter a difficulty: we don't have any target values for the hidden units. This seems to be an insurmountable problem - how could we tell the hidden units just what to do? This unsolved question was in fact the reason why neural networks fell out of favor after an initial period of high popularity in the 1950s. It took 30 years before the error backpropagation (or in short: backprop) algorithm popularized a way to train hidden units, leading to a new wave of neural network research and applications.

In principle, backprop provides a way to train networks with any number of hidden units arranged in any number of layers. (There are clear practical limits, which we will discuss later.) In fact, the network does not have to be organized in layers - any pattern of connectivity that permits a partial ordering of the nodes from input to output is allowed.

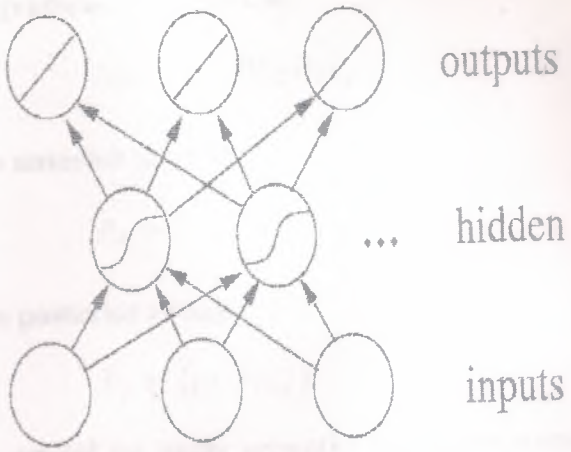

Figure 2.15. simple structure of neural network

In other words, there must be a way to order the units such that all connections go from "earlier" (closer to the input) to "later" ones (closer to the output). This is equivalent to stating that their connection pattern must not contain any cycles. Networks that respect this constraint are called feedforward networks; their connection pattern forms a directed acyclic graph or dag.

2.9.11.1. The Algorithm

We want to train a multi-layer feedforward network by gradient descent to approximate an unknown function, based on some training data consisting of pairs (**x**,**t**). The vector **x** represents a pattern of input to the network, and the vector **t** the corresponding **target** (desired output). As we have seen before, the overall gradient with respect to the entire training set is just the sum of the gradients for each pattern; in what follows we will therefore describe how to compute the gradient for just a single training pattern. As before, we will number the units, and denote the weight from unit j to unit i by w_{ij}.

Definitions:

- the **error** signal for unit j:

$$\delta_j = -\partial E / \partial net_j \qquad (2.10)$$

- the (negative) **gradient** for weight w_{ij}

$$\Delta w_{ij} = -\partial E / \partial w_{ij} \tag{2.11}$$

- the set of nodes **anterior** to unit i:

$$A_i = \{j : \exists w_{ij}\} \tag{2.12}$$

- the set of nodes **posterior** to unit j:

$$P_j = \{i : \exists w_{ij}\} \tag{2.13}$$

The gradient. As we did for linear networks before, we expand the gradient into two factors by use of the chain rule:

$$\Delta w_{ij} = -\frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} \tag{2.14}$$

The first factor is the error of unit i. The second is

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{k \in A_i} w_{ik} y_k = y_j \tag{2.15}$$

Putting the two together, we get

$$\Delta w_{ij} = \delta_i y_j \tag{2.16}$$

To compute this gradient, we thus need to know the activity and the error for all relevant nodes in the network.

Forward activaction. The activity of the input units is determined by the network's external input **x**. For all other units, the activity is propagated forward:

$$y_i = f_i(\sum_{j \in A_i} w_{ij} y_j) \tag{2.17}$$

Note that before the activity of unit i can be calculated, the activity of all its anterior nodes (forming the set A_i) must be known. Since feedforward networks do not contain cycles, there is an ordering of nodes from input to output that respects this condition.

Calculating output error. Assuming that we are using the sum-squared loss

$$E = \frac{1}{2} \sum_o (t_o - y_o)^2 \tag{2.18}$$

the error for output unit o is simply

$$\delta_o = t_o - y_o \tag{2.19}$$

Error backpropagation. For hidden units, we must propagate the error back from the output nodes (hence the name of the algorithm). Again using the chain rule, we can expand the error of a hidden unit in terms of its posterior nodes:

$$\delta_j = -\sum_{i \in P_j} \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial y_j} \frac{\partial y_j}{\partial net_j} \tag{2.20}$$

Of the three factors inside the sum, the first is just the error of node i. The second is

$$\frac{\partial net_i}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{k \in A_i} w_{ik} y_k = w_{ij} \tag{2.21}$$

while the third is the derivative of node j's activation function:

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f_j(net_j)}{\partial net_j} = f_j'(net_j) \tag{2.22}$$

For hidden units h that use the tanh activation function, we can make use of the special identity $\tanh(u)' = 1 - \tanh(u)^2$, giving us

$$f_h'(net_h) = 1 - y_h^2 \tag{2.23}$$

Putting all the pieces together we get

$$\delta_j = f_j'(net_j) \sum_{i \in P_j} \delta_i w_{ij} \tag{2.24}$$

Note that in order to calculate the error for unit j, we must first know the error of all its posterior nodes (forming the set P_j). Again, as long as there are no cycles in the network, there is an ordering of nodes from the output back to the input that respects this condition. For example, we can simply use the reverse of the order in which activity was propagated forward.

2.9.11.2. Matrix Form

For layered feedforward networks that are **fully connected** - that is, each node in a given layer connects to *every* node in the next layer - it is often more convenient to write the backprop algorithm in matrix notation rather than using more general graph form given

above. In this notation, the biases weights, net inputs, activations, and error signals for all units in a layer are combined into vectors, while all the non-bias weights from one layer to the next form a matrix W. Layers are numbered from 0 (the input layer) to L (the output layer). The backprop algorithm then looks as follows:

Initialize the input layer:

$$\vec{y}_0 = \vec{x} \tag{2.25}$$

Propagate activity forward: for l = 1, 2,..., L,

$$\vec{y}_l = f_l(W_l \vec{y}_{l-1} + \vec{b}_l) \tag{2.26}$$

where b_l is the vector of bias weights.

Calculate the error in the output layer:

$$\vec{\delta}_L = \vec{t} - \vec{y}_L \tag{2.27}$$

Backpropagate the error: for l = L-1, L-2,..., 1

$$\vec{\delta}_l = (W_{l+1}^T \vec{\delta}_{l+1}) \cdot f_l'(\vec{net}_l) \tag{2.28}$$

where T is the matrix transposition operator.

Update the weights and biases:

$$\Delta W_l = \vec{\delta}_l \vec{y}_{l-1}^T \qquad \Delta \vec{b}_l = \vec{\delta}_l \tag{2.29}$$

We can see that this notation is significantly more compact than the graph form, even though it describes exactly the same sequence of operations.

2.10. Hardwired Systems

There are some neural networks that have their connection weights predetermined for a specific problem. These weights are "hard-wired" in that they do not change once they have been determined. The most popular hardwired systems are the neural optimization networks. Neural optimization works by designing a cost function that, when minimized, solves an unconstrained optimization problem. By translating the energy function into a set of weights and bias values, the neural network becomes a parallel optimizer. Given the initial values of the problem, the network will run to a stable solution. This technique has been applied to a wide range of problems, including scheduling, routing and resource optimization.

Two other types of hardwired networks include the Avalanche Matched Filter and the Probabilistic Neural Network. These networks are considered hardwired systems because the data patterns are normalized to unit length and used as connection weights. Despite the lack of an adaptive learning procedure, each of these neural networks are very powerful in their own right.

2.11. Summary of Learning Procedures

There are several attributes of each of the neural network learning algorithms that have been described.

- Training Time - How long does it take the learning technique to adequately capture information (quick, slow, very slow, and extremely slow)?
- On-Line/Off-Line - Is the learning technique an on-line or an off-line learning algorithm?
- Supervised/unsupervised - Is the learning technique a supervised or unsupervised learning procedure? Linear/Nonlinear - Is the learning technique capable of capturing nonlinear mappings?
- Structural/Temporal Does the learning algorithm capture structural information, temporal information, or both?
- Storage Capacity is the information storage capacity good relative to the number of connections in the network?

2.12. Machine Learning

Neural networks are not the only method of learning that has been proposed for machines (although it is the most biologically related). There are a large number of machine learning procedures that have been proposed over the course of the past thirty years. Carbonell (1990) classifies machine learning into four major paradigms: "Inductive learning (e.g., acquinng concepts from sets of positive and negative examples), analytic learning (e.g., explanation-based learning and certain forms of analogical and case-based learning methods), genetic algorithms (e.g., classifier systems), and connectionist learning methods (e.g., nonrecurrent "backprop" hidden layer neural networks)." It is possible that some of the near-term applications might find it useful to combine two or more of these machine learning techniques into a coherent solution. It has only been recently that this type of approach has even been considered.

2.13. Recurrent Network

Consider the following two networks

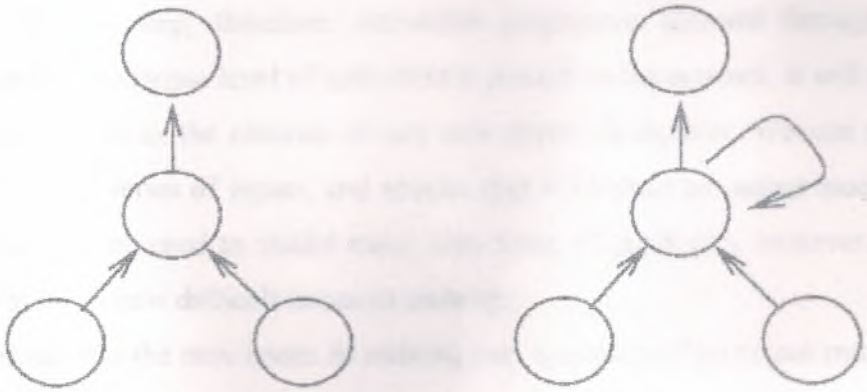

Figure 2.16. feed forward networks

The network on the left is a simple feed forward network of the kind we have already met. The right hand network has an additional connection from the hidden unit to itself. What difference could this seemingly small change to the network make?

Each time a pattern is presented, the unit computes its activation just as in a feed forward network. However its net input now contains a term which reflects the state of the network (the hidden unit activation) before the pattern was seen. When we present subsequent patterns, the hidden and output units' states will be a function of everything the network has seen so far. The network behavior is based on its history, and so we must think of pattern presentation as it happens in time.

2.13.1. Network topology

Once we allow feedback connections, our network topology becomes very free: we can connect any unit to any other, even to itself. Two of our basic requirements for computing activations and errors in the network are now violated. When computing activations, we required that before computing y_i, we had to know the activations of all units in the posterior set of nodes,

P_i. For computing errors, we required that before computing δ_j, we had to know the errors of all units in its anterior set of nodes, A_i.

For an arbitrary unit in a recurrent network, we now define its activation at time t as:

$$y_i(t) = f_i(net_i(t-1)) \qquad (2.30)$$

At each time step, therefore, activation propagates forward through one layer of connections only. Once some level of activation is present in the network, it will continue to flow around the units, even in the absence of any new input whatsoever. We can now present the network with a time series of inputs, and require that it produce an output based on this series. These networks can be used to model many new kinds of problems, however, these nets also present us with many new difficult issues in training.

Before we address the new issues in training and operation of recurrent neural networks, let us first look at some sample tasks which have been attempted (or solved) by such networks.

- **Learning formal grammars**

  Given a set of strings S, each composed of a series of symbols, identify the strings which belong to a language L. A simple example: L = $\{a^n,b^n\}$ is the language composed of strings of any number of a's, followed by the same number of b's. Strings belonging to the language include aaabbb, ab, aaaaaabbbbbb. Strings not belonging to the language include aabbb, abb, etc. A common benchmark is the language defined by the reber grammar. Strings which belong to a language L are said to be grammatical and are ungrammatical otherwise.

- **Speech recognition**

  In some of the best speech recognition systems built so far, speech is first presented as a series of spectral slices to a recurrent network. Each output of the network represents the probability of a specific phone (speech sound, e.g. /i/, /p/, etc), given both present and recent input. The probabilities are then interpreted by a Hidden Markov Model which tries to recognize the whole utterance.

- **Music composition**

  A recurrent network can be trained by presenting it with the notes of a musical score. It's task is to predict the next note. Obviously this is impossible to do perfectly, but the network learns that some notes are more likely to occur in one context than another. Training, for example, on a lot of music by J. S. Bach, we can then seed the network with a musical phrase, let it predict the next note, feed this back in as input, and repeat, generating new music. Music generated in this fashion typically sounds fairly convincing at a very local scale, i.e. within a short phrase. At a larger scale, however, the compositions wander randomly from key to key, and no global coherence arises. This is an interesting area for further work

2.13.2. The Simple Recurrent Network

One way to meet these requirements is illustrated below in a network known variously as an Elman network (after Jeff Elman, the originator), or as a Simple Recurrent Network. At each time step, a copy of the hidden layer units is made to a copy layer. Processing is done as follows:

1. Copy inputs for time t to the input units
2. Compute hidden unit activations using net input from input units and from copy layer
3. Compute output unit activations as usual
4. Copy new hidden unit activations to copy layer

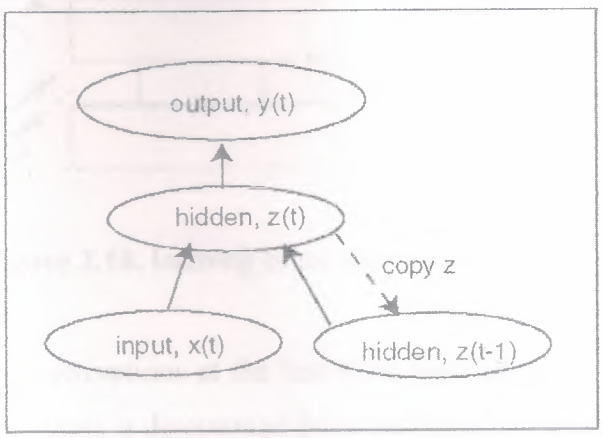

Figure 2.17. Simple recurrent neural network

In computing the activation, we have eliminated cycles, and so our requirement that the activations of all posterior nodes be known is met. Likewise, in computing errors, all trainable weights are feed forward only, so we can apply the standard backpropagation algorithm as before. The weights from the copy layer to the hidden layer play a special role in error computation. The error signal they receive comes from the hidden units, and so depends on the error at the hidden units at time t. The activations in the hidden units, however, are just the activation of the hidden units at time t-1. Thus, in training, we are considering a gradient of an error function which is determined by the activations at the present and the previous time steps.
A generalization of this approach is to copy the input and hidden unit activations for a number of previous timesteps. The more context (copy layers) we maintain, the more history we are explicitly including in our gradient computation. This approach has become known as Back Propagation Through Time. It can be seen as an approximation to the ideal of computing a gradient which takes into consideration not just the most recent inputs, but all inputs seen so far by the network. The figure below illustrates one version of the process:

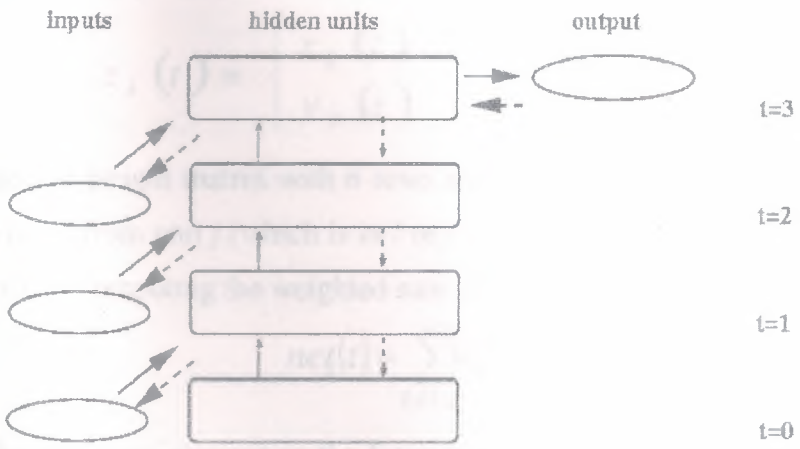

Figure 2.18. learning of recurrent neural networks

The inputs and hidden unit activations at the last three time steps are stored. The solid arrows show how each set of activations is determined from the input and hidden unit activations on the previous time step. A backward pass, illustrated by the dashed arrows, is performed to determine separate values of delta (the error of a unit with respect to its net input) for each unit and each

time step separately. Because each earlier layer is a copy of the layer one level up, we introduce the new constraint that the weights at each level be identical. Then the partial derivative of the negative error with respect to $w_{i,j}$ is simply the sum of the partials calculated for the copy of $w_{i,j}$ between each two layers.

Elman networks and their generalization, Back Propagation Through Time, both seek to approximate the computation of a gradient based on all past inputs, while retaining the standard back prop algorithm. BPTT has been used in a number of applications (e.g. ecg modeling). The main task is to to produce a particular output sequences in response to specific input sequences. The downside of BPTT is that it requires a large amount of storage, computation, and training examples in order to work well. In the next section we will see how we can compute the true temporal gradient using a method known as Real Time Recurrent Learning.

2.13.3. Real Time Recurrent Learning

In deriving a gradient-based update rule for recurrent networks, we now make network connectivity very very unconstrained. We simply suppose that we have a set of input units, $I = \{x_k(t),\ 0<k<m\}$, and a set of other units, $U = \{y_k(t),\ 0<k<n\}$, which can be hidden or output units. To index an arbitrary unit in the network we can use

$$z_k(t) = \begin{cases} x_k(t) & \text{if } k \in I \\ y_k(t) & \text{if } k \in U \end{cases} \tag{2.31}$$

Let **W** be the weight matrix with n rows and $n+m$ columns, where $w_{i,j}$ is the weight to unit i (which is in U) from unit j (which is in I or U). Units compute their activations in the now familiar way, by first computing the weighted sum of their inputs:

$$net_k(t) = \sum_{l \in U \cup I} w_{kl} z_l(t) \tag{2.32}$$

where the only new element in the formula is the introduction of the temporal index t. Units then compute some non-linear function of their net input

$$y_k(t+1) = f_k(\text{net}_k(t)) \tag{2.33}$$

Usually, both hidden and output units will have non-linear activation functions. Note that external input at time t does not influence the output of any unit until time $t+1$. The network is thus a discrete dynamical system.

Some of the units in U are output units, for which a target is defined. A target may not be defined for every single input however. For example, if we are presenting a string to the network to be classified as either grammatical or ungrammatical, we may provide a target only for the last symbol in the string. In defining an error over the outputs, therefore, we need to make the error time dependent too, so that it can be undefined (or 0) for an output unit for which no target exists at present. Let $T(t)$ be the set of indices k in U for which there exists a target value $d_k(t)$ at time t. We are forced to use the notation d_k instead of t here, as t now refers to time. Let the error at the output units be

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases} \tag{2.34}$$

and define our error function for a single time step as

$$E(\tau) = \frac{1}{2}\sum_{k \in U}\left[e_k(\tau)^2\right] \tag{2.35}$$

The error function we wish to minimize is the sum of this error over all past steps of the network

$$E_{total}(t_0, t) = \sum_{\tau=t_0+1}^{t_1} E(\tau) \tag{2.36}$$

Now, because the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the gradient for previous steps

$$\nabla_W E_{total}(t_0, t+1) = \nabla_w E_{total}(t_0, t) + \nabla_w E(t+1) \tag{2.37}$$

As a time series is presented to the network, we can accumulate the values of the gradient, or equivalently, of the weight changes. We thus keep track of the value

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}} \tag{2.38}$$

After the network has been presented with the whole series, we alter each weight w_{ij} by

$$\sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t) \tag{2.39}$$

We therefore need an algorithm that computes

$$-\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k\in U}\frac{\partial E(t)}{\partial y_k(t)}\frac{\partial y_k(t)}{\partial w_{ij}} = \sum_{k\in U} e_k(t)\frac{\partial y_k(t)}{\partial w_{ij}} \tag{2.40}$$

at each time step t. Since we know $e_k(t)$ at all times (the difference between our targets and outputs), we only need to find a way to compute the second factor.

$$\partial y_k(t) \Big/ \partial w_{ij} \tag{2.41}$$

This is given here for completeness, for those who wish perhaps to implement RTRL.

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k'(net_k(t))\left[\sum_{I\in U\cup I} w_{kI}\frac{\partial z_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t)\right] \tag{2.42}$$

where δ_{ik} is the Kronecker delta

$$\delta_{ik} = \begin{cases} 1 & \text{if i = k} \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

Because input signals do not depend on the weights in the network,

$$\frac{\partial z_I(t)}{\partial w_{ij}} = 0 \text{ for } l \in I \tag{2.44}$$

Equation becomes

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k'(net_k(t))\left[\sum_{I\in U} w_{kI}\frac{\partial y_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t)\right] \tag{2.45}$$

This is a recursive equation. That is, if we know the value of the left hand side for time 0, we can compute the value for time 1, and use that value to compute the value at time 2, etc. Because we assume that our starting state ($t = 0$) is independent of the weights, we have

$$\frac{\partial y_k(t_0)}{\partial w_{ij}} = 0 \tag{2.46}$$

These equations hold for all

$$k \in U, i \in U \text{ and } j \in U \cup I \tag{2.47}$$

We therefore need to define the values

$$P_{ij}^{k}(t) = \frac{\partial y_k(t)}{\partial w_{ij}} \tag{2.48}$$

for every time step t and all appropriate i, j and k. We start with the initial condition

$p_{ij}^{k}(t_0) = 0$

and compute at each time step

$$P_{ij}^{l}(t+1) = f_k'(net_k(t))\left[\sum_{l \in U} w_{kl} P_{ij}^{l} + \delta_{ik} z_j(t)\right] \tag{2.49}$$

The algorithm then consists of computing, at each time step t, the quantities $p_{ij}^{k}(t)$ using the above equations and then using the differences between targets and actual outputs to ompute weight changes

$$\Delta w_{ij}(t) = \mu \sum_{k \in U} e_k(t) P_{ij}^{k}(t) \tag{2.50}$$

and the overall correction to be applied to w_{ij} is given by

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t) \tag{2.51}$$

3. NEURAL LEARNING SYSTEMS FOR TECHNOLOGICAL PROCESSES CONTROL

The complexity of a number of technological processes and the pressing regime of their functioning require 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.

Taking into account the fuzziness and uncertainty of working environment of modern technological processes, an 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 the control systems. Using it allows to improve the quality of systems by paralleling computational processes and the ability for learning and adaptation which improve flexibility of systems.

In this chapter, identifications of control objects and development of direct and inverse controllers based on neural network are considered.

3.1. Modelling of Neural Control System

Assume that control object is described by the following differential equation

$$\sum_{i=1}^{n} a_{n-i} y^{(i)}(t) + c\varphi(y(t)) = \sum_{j=1}^{m} b_{m-j} u^{(j)}(t) \qquad (3.1)$$

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

The problem consists in constructing the controller for control of object (1), that would provide the target characteristic of system.

At first the development of PD-, PI-, PID- neural controllers for control of regime parameters of control object are considered. In figure 3.1 the structure of PID- neural controller is shown.

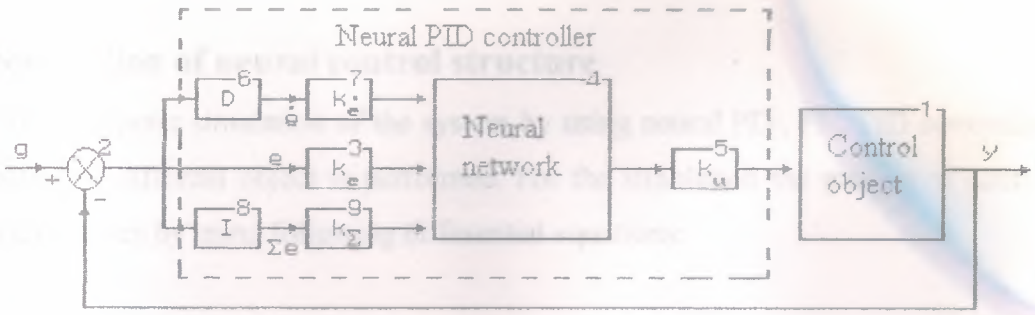

Figure 3.1. Structure of Neural PID- controller.

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 feedforward 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 a target behaviour 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 (1) would allow to achieve time response which provides target step response of the system.

The input signals error e, error derivative e' and integral value of error $\int e(t)dt$ after scaling with coefficients k_e, $\mathbf{k}_{e'}$, $\mathbf{k}_{\int e}$ are entered to neural network. The functioning of neural network is performed by using activation function U=Y/(A+|Y|). Here Y=XW.

For synthesis of neural controller the NN learning is performed by using 'backpropagation' 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.

Using learning algorithm of 'backpropagation' the values of weight coefficients of NN is found.

3.2. Simulation of neural control structure

The computer simulation of the system by using neural PD-, PI-, PID controllers for control of different object is performed. For the simulation the models of control object are chosen by using following differential equations:

$$a_0y^{(2)}(t)+a_1y^{(1)}(t)+a_2y(t)=b_0u(t) \tag{3.2}$$

where $a_{0=}$ 0.072 min^2 , a_1=0.056 min, a_2=1, b_0=60 °C/(kgf/cm^2);
here y(t)- regulation parameter of object, u(t)- neural controller's output.

$$a_0y^{(2)}(t)+a_1y^{(1)}(t)+a_2y(t)=b_0u(t-d) \tag{3.3}$$

where $a_{0=}$ 6.3 min^2 , a_1=11.2 min, a_2=1, b_0=5.1 °C/(kgf/cm^2), d=2.5 min is delay;

$$a_0y^{(3)}(t)+ a_1y^{(2)}(t)+a_2y^{(1)}(t)+a_3y(t)=b_0u(t-d) \tag{3.4}$$

where $a_{0=}$ 2.8 min^3 , a_1=3 min^2, a_2=1 min, a_3=1, b_0=34 °C/(kgf/cm^2);

The neural controllers development for given control objects are performed. In the result of learning corresponding values of neural network coefficients are determined. In fig.2(a,b,c)the time responses of PD-, PI-, PID- controllers for control object (2) are shown.

Then the results of simulation of neural controllers for technological processes control are compared with simulation results of the traditional PD-, PI-, PID-controllers. When optimal value of tuning parameters of PD- controller amplifying coefficient kp=0.08[(kgf/cm^2)/°C] and differentiaton 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 $\varepsilon_{st}(\infty) \approx 0.15x(\infty)$, 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%, settling time t=1.3 min., J=126.1.

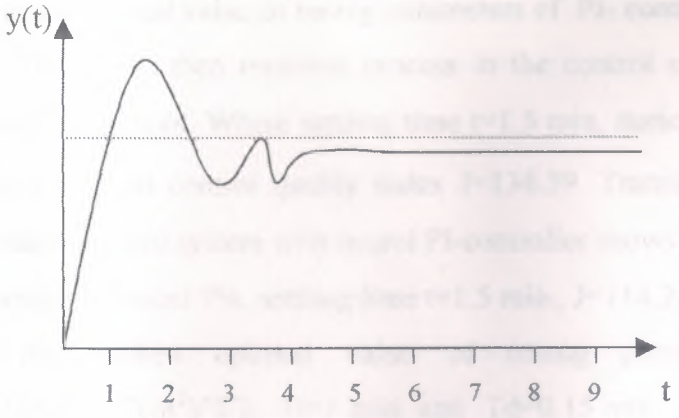

Figure 3.2.(a) PD Controller

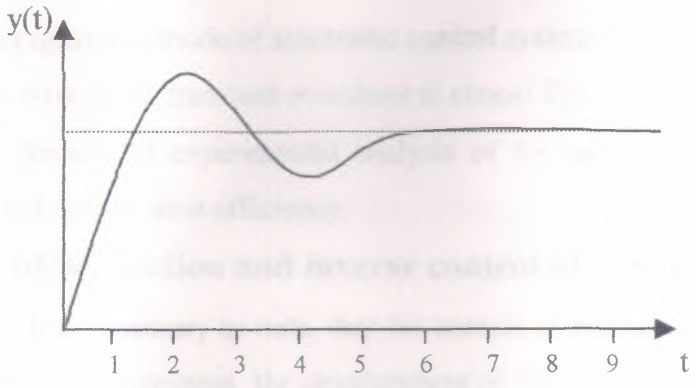

Figure 3.2.(a) PI Controller

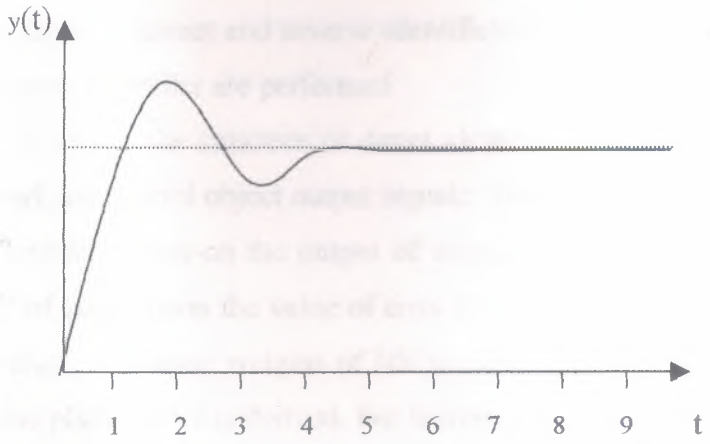

Figure 3.2.(c) PID Controller

Figure 3.2. Time response characteristics of control systems with PD, PI, PID controllers

The simulation results of comparison of traditional and neural controllers show that when optimal value of tuning parameters of PI- controller kp=0.054[(kgf/cm^2)/°C] and Ti=1 min., then transient process in the control system oscillate with 12% of transient overshoot. Where settling time t=1.5 min, static error $\varepsilon_{st}(\infty)=0$, and value of squared integral control quality index J=134.39. Transient object operation mode of automatic control system with neural PI-controller shows that static error $\varepsilon_{st}=0$, transient overshoot is almost 7%, settling time t=1.5 min., J=114.2.

Also when optimal value of tuning parameters of PID- controller kp=0.064[(kgf/cm^2)/°C], Ti=1 min. and Td=0.15 min., then transient processes in the control system oscillate with 10% of transient overshoot, settling time t=1.5 min, static error $\varepsilon_{st}(\infty)=0$, and value of squared integral control quality index J=104.75. Transient object operation mode of automatic control system with neural PID-controller show that static error $\varepsilon_{st}=0$, transient overshoot is almost 7%, settling time t=1.2 min., J=102.21.

Results of experimental analysis of the automatic control system with neural network shown their efficiency.

3.3. Identification and inverse control of dynamical systems

It is necessary to note, that for control of technological processes, functioning in the fuzzy environment, the development of fuzzy neural PD-, PI-, PID- controllers are carried out. The learning of those controllers is carried out by using α- level and interval arithmetic.

Also the direct and inverse identifications of control object (1) and development of inverse controller are performed.

In fig. 3. the structure of direct identifier is shown. 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 output. In the result of comparison the value of error $E=Y(k)-Y_N(k)$ is calculated. This error corrects the value of synaptic weights of NN to minimise error. In the result of learning on the NN the plant model is derived. For learning of NN the 'backpropagation' algorithms is used. In the NN the following activation function is used.

$$Y_N=X/(A+|X|)$$

In the inverse identification (fig.4) the input signals of NN are object output signals. Those signals enter to input of NN. After processing derived NN output signal are compared with object input U(t) and the value of error $E(k)=U(k)-U_N(k)$ is

calculated. Using above mentioned learning algorithm the correction of weight coefficients is performed. 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.

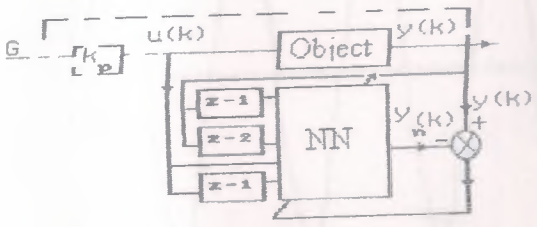

Figure 3.3 Structure of direct identification

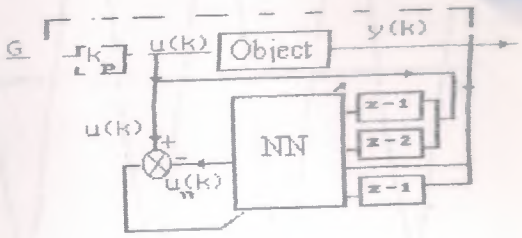

Figure 3.4. Structure of inverse Identification

The program performing direct, inverse identification processes and controlling object is developed. The system is implemented using Turbo_Pascal and a computer IBM PC/AT.

The results of direct and inverse identification processes of the plant are shown in figure 5(a,b). During the identification the sinusoidal signal is given to the input of the system. In the figure the straight line shows the obect output (3.5a) and object input (3.5b) and dotted line shows output of the neural identifiers. As shown in the figures the input and output of the object coincided with neural identifiers. This confirmed the adequacy of the derived models.

Results of inverse identification are used for development of a neural controller for control of object. Fig. 6 shows the structure of the controller.

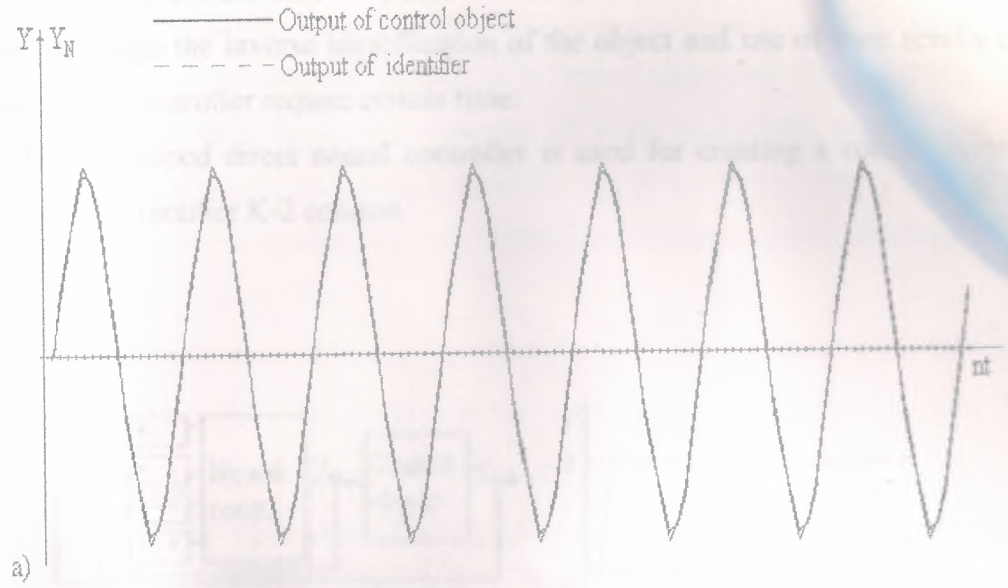

Figure 3.5(a) Simulation results of direct identification

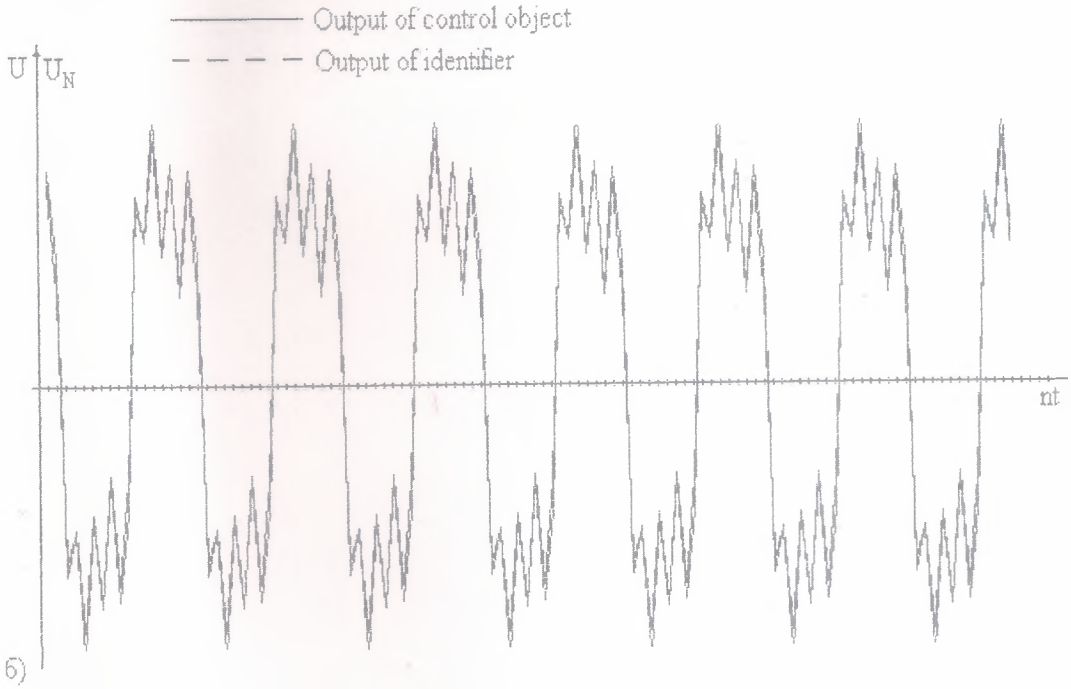

Figure 3.5(b) Simulation results of inverse identification

Also in fig. 3.6 the 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.

The developed direct neural controller is used for creating a control system of temperature of rectifier K-2 column.

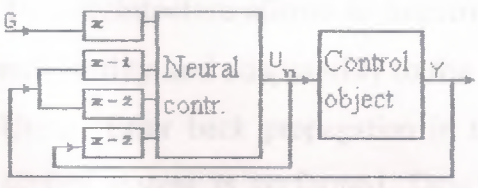

Figure 3.5. Structure of inverse controller

Figure 3.6 Time response of control system system

CONCLUSION

The analyses of some technological processes show that they are characterized by uncertainty of their work, non-linearity of their functioning principles. Use of traditional technology is not enough describe those processes. For this reason the construction of control system based of artificial intelligence methods such as neural network technology is proposed.

The architecture of neural control system for technological process is given. This architecture allows to improve accuracy of the control system due to its learning ability and adaptability to the changing of environment.

Using "Error back propagation in time" learning algorithm, the synthesis of neural control system is performed. Developed control system allows to get high dynamical accuracy of the system to changing environmental conditions, that show considerable advantage of those system over traditional control systems.

The simulation of neural control system is performed in Turbo Pascal language. Modeling and simulation of developed control system show the efficiency of proposed technology and programming tools in the thesis.

REFERENCES

[1] Y.H.Pao, *A computational intelligence perspective on process monitoring and optimization,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[2] L.MOTUS, *Trends in artificial intelligence applications for real-tme control,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[3] S.FATIKOW, K.SUNDERMAN, Neural-*based learning in grasp force control of a robot hand,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[4] P.Rutherford, B. Lennox, G.A. Montague, *Case studies in process modelling and condition monitoring using artificial neural networks,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[5] R.A. Vingerhoeds, A.J.Krijgsman, *The nectar-project research into the application of neural networks for flight control,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[6] G.Schram, L.Karsten, B.J.A.Krose, *Optimal attitude control of satellites by artificial neural networks: a pilot study,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[7] J.F.Peters, L.Baumela, D.Maravall, S.Ramanna, *Logical design of neural controllers,* IFAC Artifical Intelligence in Real Time Control, Valencia, Spain, 1994

[8] I.H.Muratov, R.H.Abiev, R.R.Aliev, *Neural learning systems for technological processes control,*Azerbaijan State Oil Academy, Dep. Automatic Control System Azadlig av.20, Baku, Azerbaijan

[9] R.A.Aliev, R.H.Abiev, R.R.Aliev. Synthesis of automatic control system by learned neural network fuzzy controller. Texnicheskaya kibernetika, Moskva, N3,1994. (Russian)

[10] R.H.Abiev. Controllers based on neural networks//Uchenie zapiski, AzGNA,1994,88-97pp. (Russian)

[10] R.H.Abiev, K.W.Bonfig, F.T.Aliev. Controller based on fuzzy neural network for control of technological process.//ICAFS-96, Siegen, Germany, June 25-27.

[11] R.H.Abiev, R.R.Aliev. Neural controller for control technological processes /Uchenie zapiski,N1, AzGNA,1995,147-152pp. (Russian)

[12] U.K.Korgman, *Introduction to neural computing and categories of N.N. applications to Guidance, Navigation and Control,* AGARD, 1994

[13] P.K. Simpson, *Neural Network Paradigms,* AGARD, 1994

[14] Richard J. Mammone, Y.Zeevi, *Neural Networks Theory and Applications,* Acadmic Press, USA, 1991

[15] M. Glesner, W.Pochmuller, *Neuro Computers*, Chapman and Hall, UK, 1994

[16] Robert L. Harvey, *Neural Network Principles,* Prentice Hall Inc., NJ, 1994