NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

INTELLECTUAL CONTROL SYSTEM FOR TECHNOLOGICAL PROCESSES

GRADUATION PROJECT
COM – 400

Student: Mohammed Alhaj Hussein (991288)

Supervisor: Assoc.Prof.Dr Rahib ABIYEV

Nicosia - 2002

ACKNOWLDGMENT

First of all I would like to thank Assoc. Prof. Dr. Rahib Abiyev for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

I would like to thank all of my friends who helped me to overcome my project especially Yousef, and manna.

Finally, I would like to thank my family, especially my parents. Their love and guidance saw me through doubtful times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together.

They have made countless sacrifices for my betterment. I can't repay them, but I do hope that their endless efforts will bear fruit and that I may lead them, myself and all who surround me to a better future.

ABSTRACT

Human beings epitomize the concept of "intelligent control." Despite its apparent computational advantage over humans, no machine or computer has come close to achieving the level of sensor-based control which humans are capable of. Thus, there is a clear need to develop computational methods which can abstract human decision-making processes based on sensory feedback.

Neural networks offer one such method with their ability to map complex nonlinear functions.

The aim of graduating project is the development of neural control system for technological processes. To achieve this aim the application problem of neural system for technological processes is considered .The model of neural systems, their architectures and learning algorithms are given.

Using neural structure the development of the neural control system is preformed, Controller is constructed on the base of neural network. The learning algorithm of neural network for controllers is described.

The modeling of the neural identification and control system is performed, Results of simulations of the developed and the traditional control system showed the improved time response characteristics of previous.

TABLE OF CONTENTS

INTRODUCTION

Researchers in the field of robotics and autonomous systems frequently find themselves looking towards human intelligence as a guide for developing "intelligent" machines. Paradoxically, control tasks, which seem easy or even trivial for humans, are often extremely difficult or impossible for computers or robots to duplicate. Rule-based systems usually fail to anticipate every eventuality and thus are ill suited for robots in uncertain and new environments. There is a clear need to develop computational methods which can, in a general framework, abstract the human decision-making process based on sensory feedback.

Modeling and identifying human control processes can be a significant step towards transferring human knowledge and skill in real-time control. This can lead to more intelligent control systems, and can bridge the gap between traditional artificial intelligence theory and the development of intelligent machines.

Artificial neural networks have shown great promise in identifying complex nonlinear systems. Thus, neural networks are well suited for generating the complex internal mapping from sensory inputs to control actions, which humans possess. Our goal is to develop a feasible neural network-based method for identifying human control strategy and transferring that control strategy to control systems. To this end, we are looking at an efficient and flexible neural network architecture that is capable of modeling nonlinear dynamic systems.

The project consists of introduction, 4 chapters and conclusion.

Chapter1 describes the states of neural control system however, its describes the two problems. First, the neural control of intelligent structures and the second, autonomous vehicle navigation.

Chapter2 describes the architecture of neural control systems for technological process, including the structure of neural system and descriptions of the functions of its main block are given. The neural network structures and their operation principles considering some problems, also the description of the learning in neural network has been considered, and some historical background of neural network has considered too.

Chapter3 describes the development of neural control system for technological process. The desired time response characteristic of system, neural control system's learning algorithm and characteristic of technological process are described. Using these the synthesis of procedures and simulation of neural control system are performed.

Chapter 4 describes the provide background information on this new architecture for neural network learning and a theoretical basis for its use. Then simulation results presented for this architecture in identifying both static and dynamic systems, including a nonlinear controller for an inverted pendulum system. Finally, some preliminary results in modeling human control strategy have been showed and discussed.

Conclusion presents the important obtained result that the project discussed and contributes in the project itself.

CHAPTER ONE. STATE OF APPLICATION PROBLEMS OF NEURAL NETWORK FOR TECHNOLOGICAL PROCESSES

1.1. Neural Control of Intelligent Structure

Smart structures as envisioned with embedded and distributed sensor/actor devices impose new challenges and problems for control engineering. The reasons for this are twofold: First, the design of embedded and distributed sensor/actor devices gives rise to new questions about the development of more appropriate control strategies interpreting the global versus local control strategy trade-off from a new and "distributed" perspective. Second, due to the non-linearity of the system components it is often too difficult to derive a system model of the smart structure suitable for classical controller design based on an exact analytical model using first principles.

Neural architectures such as neural networks offer in these cases the advantage to avoid the analytical modelling of smart structures and to "learn" the system transfer function from available experimental or simulated data instead. The work described here is focusing on the learning aspect of smart structure controllers with neural architectures and is organized along the following two main research directions of the basic research effort that aims at the development of novel neural control architectures. In this respect it has been aimed to resolve the "black-box-character" in neural network applications to allow a deeper mathematical analysis of the neural network after training.

This goal has been achieved through the introduction of the concept of dimensional homogeneity for neural networks [6], which leads to the emergence of dimensionless similarity parameters in the neural nets and allows to interpret the neural mapping in the network as the similarity function of the physical object under consideration, and, through the identification of the neural correspondence for classical control engineering techniques such as the Laplace-Transform [7]. It is expected that these two developments will ease a future performance analysis and a more direct comparison of classical controllers with neural control approaches including a future stability proof for neural control.

The practical development and design of novel controllers with neural architectures for different reference models [1,2,7]. These reference models are:

1) The tether deployment for small capsule re-entry,

2) The generic bump panel,

3) The adaptive helicopter blade,

4) The acoustic cavity.

In the practical development of neural controllers for these applications, the pre-processing of the training data, the employed training procedures for the neural network controller, the control performance and accuracy have been investigated. A generic procedure for the design of neural controllers has also been established for these purposes.

These two main research directions characterize the research results of the project A1 "Neural control of Intelligent Structures" which have been achieved in cooperation with other projects in the framework of the collaborative research project SFB 409 "Smart Structures in Aerospace Engineering". The details about the above mentioned different system models to be controlled and the simulation or experimental data have been provided from the partner project, while the neural modelling and the neural controller design has been performed in the project A1. The lessons learned and the results obtained are described in the following.

Two different neural network control schemes, a direct and an indirect control scheme, have been proposed in the literature [5]. For a detailed overview of neural control methodologies see [4]. While the direct neural control scheme in figure .1.1 doesn't use a model of the plant and is known to suffer from stability problems, the indirect neural control scheme makes use of a previously identified neural plant model, see figure1.2.

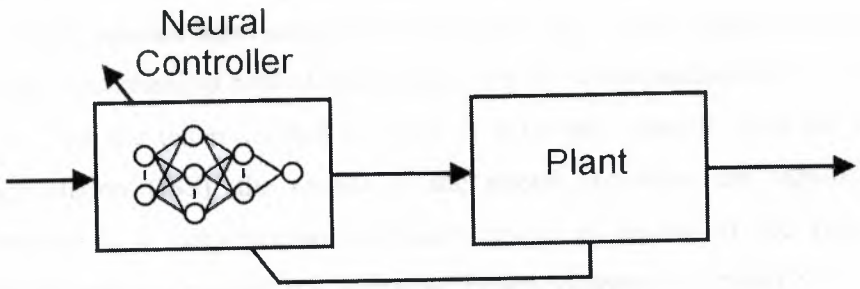

Figure .1.1. The direct neural control scheme

The neural plant model in figure 1.2 is trained using the squared error between the actual plant output and the model. Training is ceased when the approximation of the plant is good enough and the neural network can be used as a plant model for the

training of the controller without the use of the actual system. After successful plant identification, the neural controller is trained with an inverse training scheme [4] as shown in figure1.2. The control input is fed into the plant and the neural plant model.

The error between the commanded input and the plant output and the neural network output is then propagated back through the neural plant model using the first steps of the well-known Back propagation algorithm. The error found for the input neuron of the neural plant model corresponds directly to the error of the output neuron of the neural controller and can be used for the training with standard learning algorithms

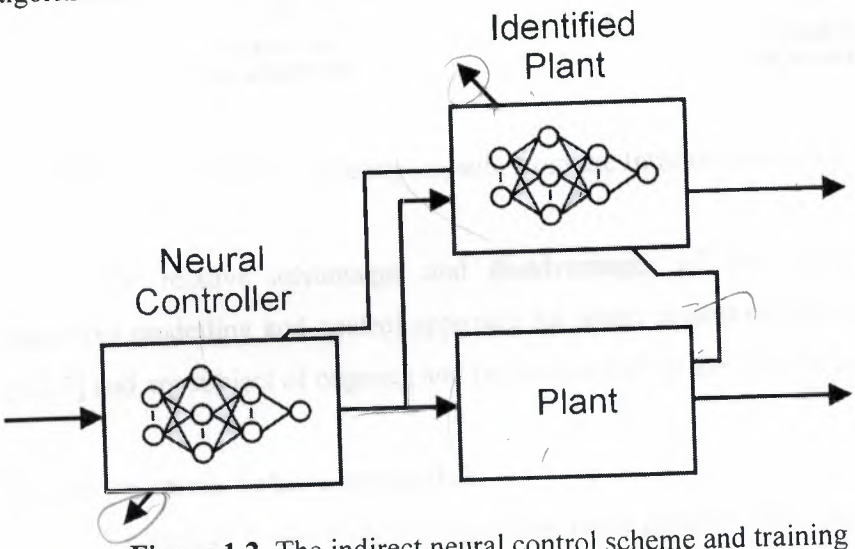

Figure 1.2. The indirect neural control scheme and training

The neural network controller is usually structured according to the neural network plant model using external feedback of the control signal and delayed values of the commanded input using time-delay lines. This neural control approach has been compared to classical controller designs using the mentioned reference examples.

For the tether-assisted de orbit of a re-entry capsule from the international space station (ISS), the results of the neural controller are significantly better compared to a conventional controller design as shown in the figure1.3.below, Together with the project B3 "Adaptive Tether Systems for Orbital Systems", a time-variant neural network controller has been developed for the deployment of a tethered re-entry capsule from the International Space Station (ISS).

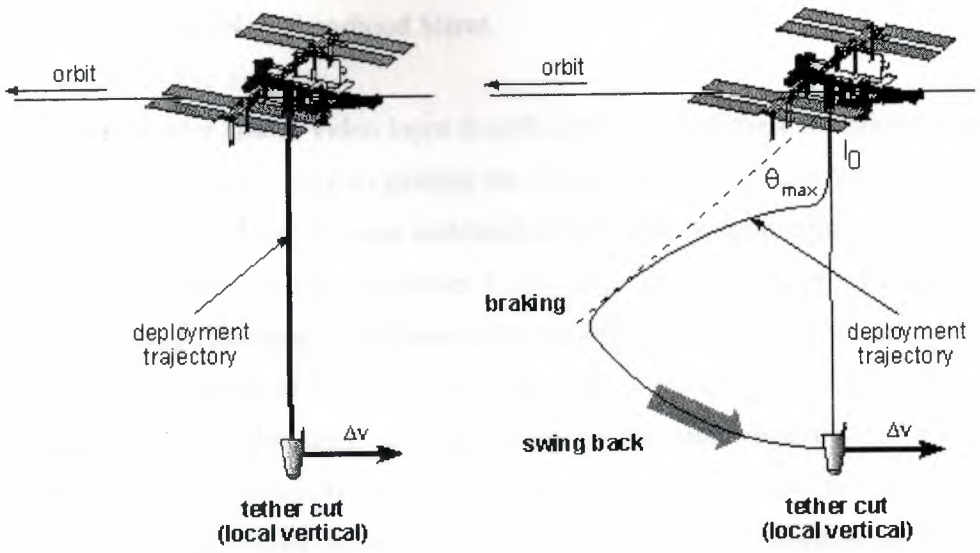

Figure 1.3. Tethered re-entry capsule from the International Space Station (ISS).

The relative advantages and disadvantages of the inductive versus the deductive modelling and control approach for smart structures have been reported in [1,2,7] and are subject of ongoing and future research in the SFB 409 project A1.

1.2. Autonomous Vehicle Navigation

Vision-based autonomous vehicle and robot guidance have proven difficult for algorithm-based computer vision methods, mainly because of the diversity of unexpected cases that must be explicitly dealt with in the algorithms and the real-time constraint, Pomerleau successfully demonstrated the potential of neural networks for overcoming these difficulties, his ALVINN (Autonomous Land Vehicle in Neural Networks) set a world record for autonomous navigation distance. After training on a two-mile stretch of highway, it drove the CMU Navlab. Equipped with video cameras and laser range sensors, for 21.2 miles with an average speed of 55 mph on a relatively old highway open to normal traffic, ALVINN was not disturbed by passing cars while it was being driven autonomously, ALVINN nearly doubled the previous distance world record for autonomous navigation. What is surprising is the simplicity of the networks and the training techniques used in ALVINN, which consists of several networks, each trained for a specific road situation:

1) Single-lane paved road.
2) Single-lane dirt road.

3) Two-lane Neighbourhood Street.

4) Multilane highway.

A monocular colour video input is sufficient for all of these situations; therefore, no depth perception is used in guiding the vehicle. Not using stereovision saves a significant amount of time, because matching of correspondence points in a stereo pair of images is computationally expensive. Laser rangefinder and laser reflectance inputs are also tested. The laser reflectance input resembles a black-and-white video image and can be handled in the same way as a video image. Reflectance input is advantageous over video input, because it appears the same regardless of the lighting conditions. This allows ALVINN to be trained in daylight and tested in darkness. Laser rangefinder input is useful for obstacle avoidance. However, a laser range image needs to be processed differently, because its image pixel values represent distance instead of lightness. We will focus the discussion on video image input.

A network in ALVINN for each situation consists of a single hidden layer of only four units, an output layer of 30 units and a 30 X 32 retina for the 960 possible input variables. The retina is fully connected to the hidden layer, and the hidden layer is fully connected to the output layer, as shown in figure1.4 for two representative nodes (out of a total of 960). The graph of the feed forward network is a node-coalesced cascade of directed versions of bipartite graphs K960, 4 and k4, 30 Pomerleau tried networks with more layers and more hidden units but did not observe significant performance improvement over this simple network, Because of the real-time constraint of the task, a simple network is definitely preferred.

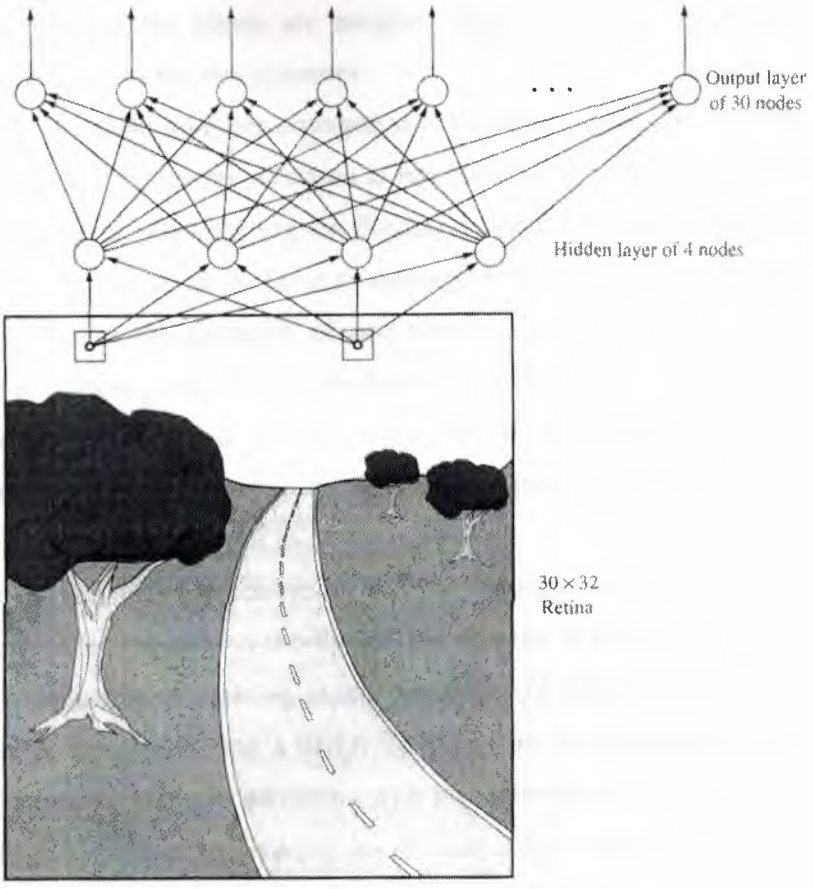

Figure 1.4. The graph of a network in ALVINN.

It is a node-coalesced of two directed bipartite graphs, The Image on the retina is a low-resolution version of a cooler video image with 480 X 512 pixels. A 16X16 neighbourhood in the video image is randomly sampled and averaged to produce a single pixel on the retina .the outputs from the three channels of a colour video image-namely, red(R), green (G), and blue (B) are combined to produce

$$P = \frac{B}{255} + \frac{B}{R+G+B}$$

Where, P is the brightness of the combined image. This combination is based on empirical observation. What is interesting is that it approximates the learning result if one chooses to add another layer to learn the pre-processing from video image to the retina. The darkest 5% of the pixels on the retina are assigned the minimum activation level of -1, and the brightest 5% are assigned the maximum activation level of 1. The

remaining 90% of the pixels are assigned activation values proportional to their brightness relative to the two extremes.

The 30 output units are arranged in a one-dimensional array for controlling the steering wheel. The steering direction is represented by a Gaussian activation pattern in the output layer, illustrated in the Figure1.5; the distributed pattern representation of the output proves to be useful in evaluating the reliability of the network output. If the vehicle under the guidance of one network (e.g., for single-lane paved road) transits into a new situation (e.g. multilane highway) the network will be confused. There is a high likelihood that the output pattern will significantly deviate from a Gaussian pattern. This signals; the ALVINN to pick another network to guide the vehicle.

Each network is trained using the back propagation algorithm with a technique Pomerleau called training-on-the-fly: i.e. the network is trained by observing a person driving a sequence of training pairs, consisting of input images and the person's response, is obtained during a drive. Training can be performed at the same time. There are several potential problems with the training-on-the-fly approach. They are all due to the low level of diversity or- in other words, the high level of similarity, in the training data. For example, the network needs to learn how to recover from Various mistakes, a sequence of consistently similar training data will also cause the network to over learn the current situation and forget about what it might have learned about other situations, Diversity in the training data is necessary for valid generalization. Pomerleau used several techniques to solve these problems.

First, the inner images and the steering directions are geometrically transformed as if the vehicle had been in different positions relative to the road. Second, structured noise is added to the input images to simulate different situations on the road, such as passing cars. Guardrails, and trees. New training pairs are formed using a new image with added structure noise and the same steering direction as the noise-free image. These techniques greatly increase the diversity of the training data. Thus leading to good generalisation of learning.

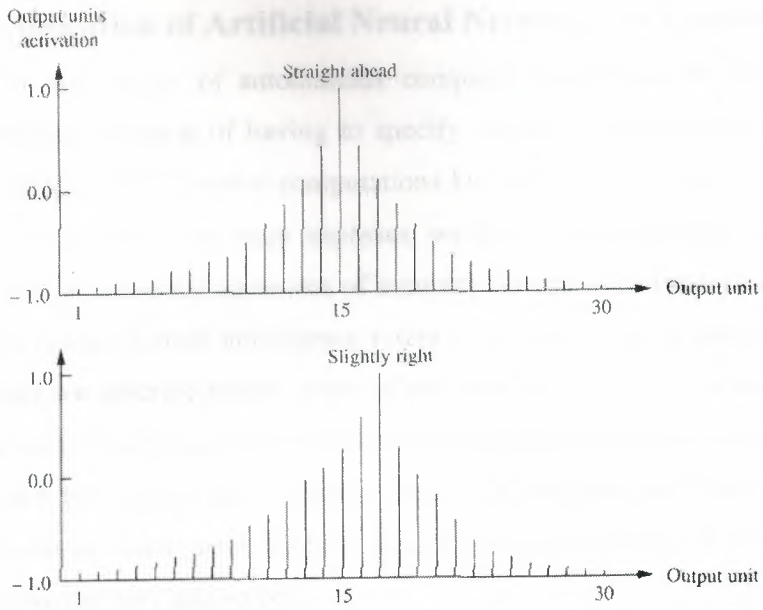

figure1.5. Representation of two steering directions in the output layer from the plot of output unit activation values versus the output unit.

After each network is trained for a specific situation, how is the system going to decide when to use which network? Pomerleau proposed two techniques for selecting a network: output appearance and input reconstruction. Output appearance measures the deviation of the shape of the output response from the Gaussian shape. Although not always true, there is a high level of correlation between the Gaussian shape of the output response and the applicability of the network to the current situation. Input reconstruction feeds the output response back through the connections and the hidden layer to reconstruct an input image. The difference between the real input image and the reconstructed image provides another indication of the applicability of the network to the current situation. These two techniques can then be used to guide the choice of the right network for the current situation, Due to the Fact that neural networks are not good at remembering maps and planning the route, a symbolic component is added to the system for these functions. The symbolic component is also responsible for generating structured noises and transformations to increase the diversity of the training data and for coordinating all the components in the system.

1.3. Application of Artificial Neural Network 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 suspect of being so. To circumvent such impasses, we resort to complexity reducing strategies and tactics, which trade some loss of accuracy for significant reduction in complexity; the term computational intelligence refers to such complexity reduction methods. In this paper we describe briefly some of our own work in this area and then develop a computation intelligence view of the tasks of process monitoring and optimization, as performed by autonomous system. Some important current fields of discovery in computational intelligence include neural net computing evolutionary, fuzzy sets, associative memory and so on.

Some of the theory bounded evolutional trends in real time al application are pointed out based system. The evolutional main stream is increasing interdisciplinary integration.

Three sub trends are illustrated on examples: mechanical combination of method, 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 system designed 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. The most attention is given to the neural based force adaptation level implemented by three-layered back propagation neural network. A computer based simulation system for the big_in_hole insertion task is developed to analyse the capabilities of the neural controllers. Their behaviour 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 of the UK has supported the transfer of neural technology as part of the campaign, the university of the new castle and EDS advanced technologies group have set-up a process monitoring and control club.

The logical design of a neural controller is a achieved by representing a neural computation as a stochastic timed linear proof with a built-in system for rewards and punishments based on the timelines of a computation performed by a neural controller.

Logical designs are represented with stochastic forms of proof nets and proof boxes. Sample application of the logical design methodology of the truck-backer upper and a real time object recognition and tracking system (RTorts) are presented. Performance result of the implementation module of the (RTorts) are given and compared to similar system.

The work describe in the neural network of intelligent structure is focusing on the learning aspect of smart structure controllers with neural architectures along the following two main research directions of The basic research effort that aims at the development of novel neural control architectures. So it used two different neural network control schemes, a direct and an indirect control scheme.

For the autonomous navigation concept is explaining a network in ALVININ for each situation consists of a single hidden layer of only four units. The retina is fully connected to the hidden layer as well as hidden layer is fully connected to the output layer as a result of the real time constraint of the task, a simple network is definitely preferred.

Each network is trained using the back propagation algorithm with a technique Pomerleau called training-on-the-fly; training can be performed at the same time.

Pomerleau proposed two techniques for selecting a network: output appearance and input reconstruction as we have declared before, this two techniques can then be used to guide the choice of the right network for the current situation.

CHAPTER TWO. STRUCTURE AND LEARNING OF NEURAL NETWORKS

2.1. Introduction To Neural Networks

The power and speed of modern digital computers is truly astounding. No human can ever hope to compute a million operations a second. However, there are some tasks for which even the most powerful computers cannot compete with the human brain, perhaps not even with the intelligence of an earthworm.

Imagine the power of the machine, which has the abilities of both computers and humans. It would be the most remarkable thing ever. And all humans can live happily ever after. This is the aim of artificial intelligence in general.

When we are talking about a neural network, we should more properly say "artificial neural network" (ANN), because that is what we mean most of the time. Biological neural networks are much more complicated than the mathematical models we use for ANNs. But it is customary to be lazy and drop the "A" or the "artificial".

An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information.

The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

At the core of neural computation are the concepts of distributed, adaptive, and non-linear computing. Neural networks perform computation in a very different way than conventional computers, where a single central processing unit sequentially dictates every piece of the action.

Evolving from neuro-biological insights, neural network technology gives a computer system an amazing capacity to actually learn from input data. Artificial neural networks have provided solutions to problems normally requiring human observation and thought processes. Some real world applications include:

- Quality Control
- Financial Forecasting
- Economic Forecasting
- Credit Rating
- Speech & Pattern Recognition
- Biomedical Instrumentation
- Process Modelling & Management
- Laboratory Research
- Oil & Gas Exploration
- Health Care Cost Reduction
- Targeted Marketing
- Defence Contracting
- Bankruptcy Prediction
- Machine Diagnostics
- Securities Trading

2.2 Some Other Definitions of a Neural Networks

According to the DARPA Neural Network Study (1988, AFCEA International Press, p. 60): a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

According to Haykin, S. (1994), Neural Networks: A Comprehensive Foundation, NY: Macmillan, p. 2:

A neural network is a massively parallel-distributed processor that has a natural propensity for storing experiential knowledge and making it available for use, It resembles the brain in two respects:

1.The network through a learning process acquires knowledge.

2.Interneuron connection strengths known as synaptic weights are used to store the knowledge.

ANNs have been applied to an increasing number of real-world problems of considerable complexity. Their most important advantage is in solving problems that are too complex for conventional technologies problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found.

ANNs have been applied to an increasing number of real-world problems of considerable complexity. Their most important advantage is in solving problems that are too complex for conventional technologies problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found. In general, because of their abstraction from the biological brain, ANNs are well suited to problems that people are good at solving, but for which computers are not. This problem includes pattern recognition and forecasting (which requires the recognition of trends in data).

Neural Networks approaches this problem by trying to mimic the structure and function of our nervous system. Many researchers believe that AI (Artificial Intelligence) and neural networks are completely opposite in their approach. Conventional AI is based on the symbol system hypothesis. Loosely speaking, a symbol system consists of indivisible entities called symbols, which can form more complex entities, by simple rules. The hypothesis then states that such a system is capable of and is necessary for intelligence.

The general belief is that Neural Networks is a sub-symbolic science. Before symbols themselves are recognized, some thing must be done so that conventional AI can then manipulate those symbols. To make this point clear, consider symbols such as cow, grass, house etc. Once these symbols and the "simple rules" which govern them are known, conventional AI can perform miracles. But to discover that something is a cow is not trivial. It can perhaps be done using conventional AI and symbols such as - white, legs, etc. But it would be tedious and certainly, when you see a cow, you instantly recognize it to be so, without counting its legs.

But this belief that AI and Neural Networks are completely opposite is not valid because, even when you recognize a cow, it is because of certain properties, which you observe, that you conclude that it is a cow. This happens instantly because various parts of the brain function in parallel. All the properties, which you observe, are "summed up". Certainly there are symbols here and rules - "summing up". The only difference is that in AI, symbols are strictly indivisible, whereas here, the symbols (properties) may occur with varying degrees or intensities.

Only breaking this line of distinction between AI and Neural Networks, and combining the results obtained in both, towards a unified framework, can make progress in this area.

2.3. Biological Information Process

To imitate biological information processing models for different level 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 behaviour of the neuron. On the network level The interconnection of identical neurons to form network is examined to describe specific sensor and motor city-related functions such as filtering, projection operations, controller function. In non-linear, biological system, network 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.

2.3.1 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 non-linear 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.

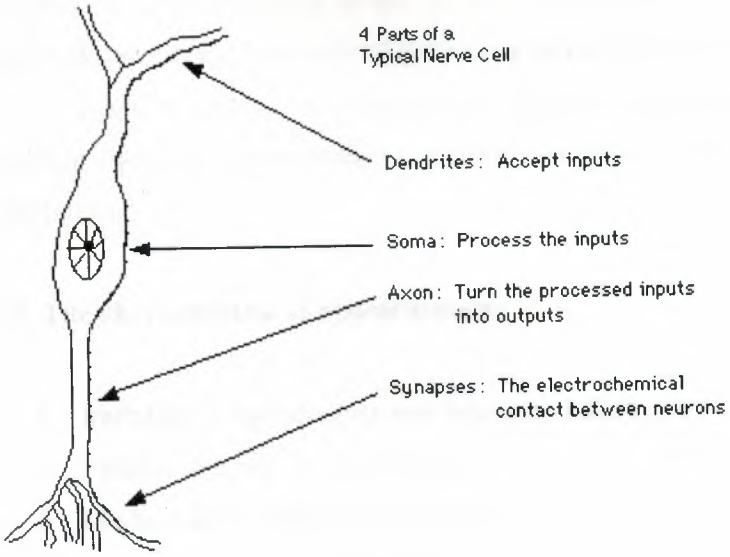

figure2.1. Biological Neuron

2.3.2 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.

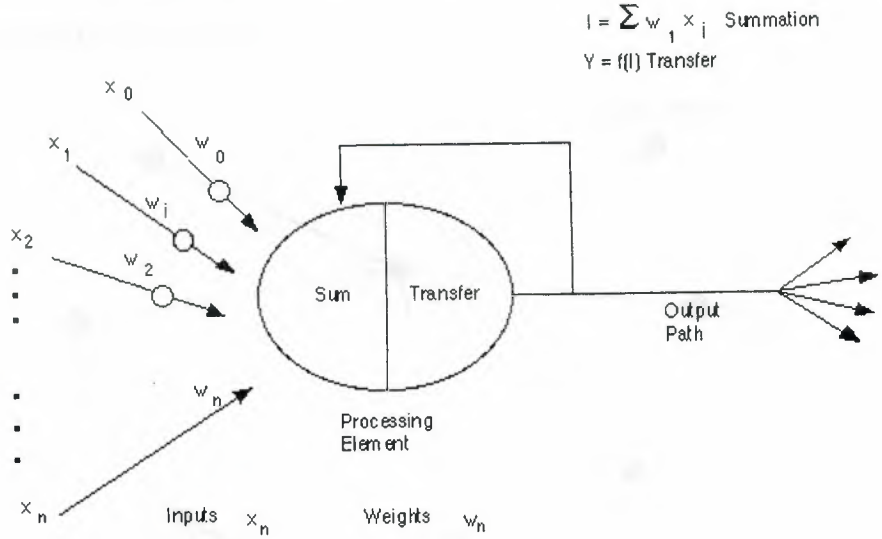

Figure2.2. Artificial Neuron

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.

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.

2.4. The characteristic of neural systems

1. Imitation of the structure and function of the brain.
2. Parallel information processing.
3. Implicit knowledge representation.
4. Application of inductive reasoning.
5. Learning occurs within the system.

2.5. The Structure of the Nervous System

For our purpose, it will be sufficient to know that the nervous system consists of neurons, which are connected to each other in a rather complex way. Each neuron can be thought of as a node and the interconnections between them are edges as shown below in the figure2.3:

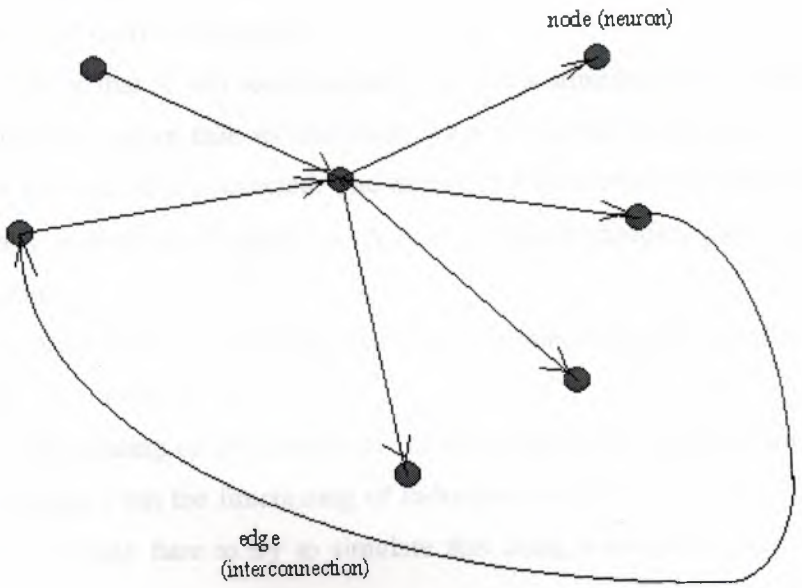

figure2.3. Edge interconnections.

Such a structure is called as a directed graph. Further, each edge has a weight associated with it, which represents how much the two neurons, which are connected by it, can interact. If the weight is more, then the two neurons can interact much more a stronger signal can pass through the edge.

2.6. Functioning of the Nervous System

The nature of interconnections between 2 neurons can be such that one neuron can either stimulate or inhibit the other. An interaction can take place only if there is an edge between 2 neurons. If neuron A is connected to neuron B as below with a weight w, in the figure1.2

Figure2.4. The Edge Between Two Neurons.

Then if A is stimulated sufficiently, it sends a signal to B. The signal depends on the weight w, and the nature of the signal, whether it is stimulating or inhibiting. This depends on whether w is positive or negative. If sufficiently strong signals are sent, B may become stimulated.

Note that A will send a signal only if it is stimulated sufficiently, that is, if its stimulation is more than its threshold. Also if it sends a signal, it will send it to all nodes to which it is connected. The threshold for different neurons may be different. If many neurons send signals to A, the combined stimulus may be more than the threshold.

Next if B is stimulated sufficiently, it may trigger a signal to all neurons to which it is connected.

Depending on the complexity of the structure, the overall functioning may be very complex but the functioning of individual neurons is as simple as this. Because of this we may dare to try to simulate this using software or even special purpose hardware.

2.7 The Difficulty of Modelling a Brain-like Neural Network

We have seen that the functioning of individual neurons is quite simple. Then why is it difficult to achieve our goal of combining the abilities of computers and humans?

The difficulty arises because of the following:

It is difficult to find out which neurons should be connected to which. This is the problem of determining the neural network structure. Further, the interconnections in the brain are constantly changing. The initial interconnections seem to be largely governed by genetic factors.

The weights on the edges and thresholds in the nodes are constantly changing. This problem has been the subject of much research and has been solved to a large extent. The approach has been as follows: Given some input, if the neural network makes an error, then it can be determined exactly which neurons were active before the error. Then we can change the weights and thresholds appropriately to reduce this error.

For this approach to work, the neural network must "know" that it has made a mistake. In real life, the mistake usually becomes obvious only after a long time. This situation is more difficult to handle since we may not know which input led to the error.

Also notice that this problem can be considered as a generalization of the previous

problem of determining the neural network structure. If this is solved, that is also solved. This is because if the weight between two neurons is zero then, it is as good as the two neurons not being connected at all. So if we can figure out the weights properly, then the structure becomes known. But there may be better methods of determining the structure.

The functioning of individual neurons may not be so simple after all. For example, remember that if a neuron receives signals from many neighbouring neurons, the combined stimulus may exceed its threshold. Actually, the neuron need not receive all signals at exactly the same time, but must receive them all in a short time-interval.

It is usually assumed that such details will not affect the functioning of the simulated neural network much. But may be it will.

Another example of deviation from normal functioning is that some edges can transmit signals in both directions. Actually, all edges can transmit in both directions, but usually they transmit in only 1 direction, from one neuron to another.

2.8.Neural Network Topologies

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

Neural networks are built from a large number of very simple processing elements that individually deal with pieces of a big problem. A processing element (PE) simply multiplies an input by a set of weights, and a nonlinearly transforms the result into an output value. The principles of computation at the PE level are deceptively simple. The power of neural computation comes from the massive interconnection among the PEs, which share the load of the overall processing task, and from the adaptive nature of the parameters (weights) that interconnect the PEs.

Normally, a neural network will have several layers of PEs. The most basic and commonly used neural network architecture is the multi layer perceptron (MLP). The diagram (figure 2.5.) below illustrates a simple MLP. The circles are the PEs arranged in layers. The left row is the input layer, the middle row is the hidden layer, and the right row is the output layer. The lines represent weighted connections (i.e., a scaling factor) between PEs.

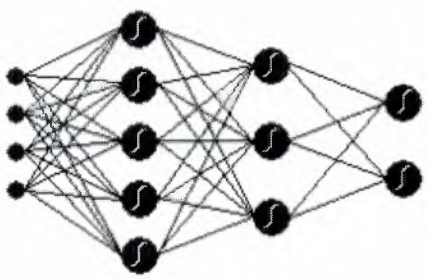

figure2.5. A simple Multi Layer Perceptron

The performance of an MLP is measured in terms of a desired signal and an error criterion. The output of the network is compared with a desired response to

produce an error. An algorithm called back propagation is used to adjust the weights a small amount at a time in a way that reduces the error. The network is trained by repeating this process many times. The goal of the training is to reach an optimal solution based on the performance measurement.

We shall now try to understand different types of neural networks

2.8.1. 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.

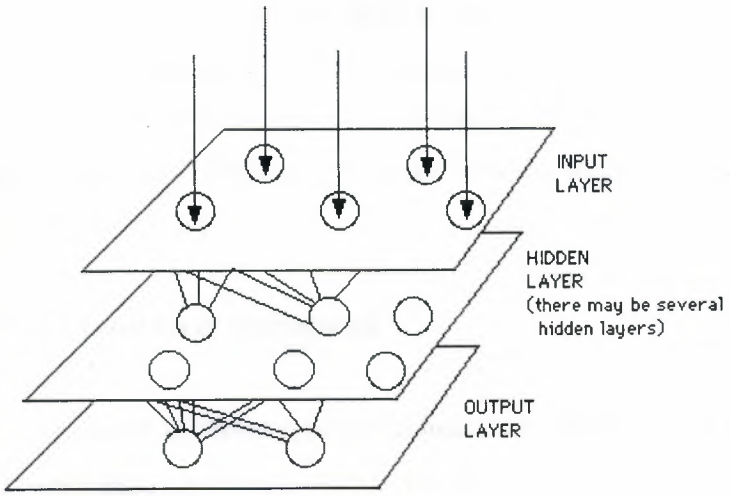

figure2.6. Layer Structure

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.

2.8.2. 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

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.

2.8.2.1 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.

2.8.2.2 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-centre/off surround** A neuron within a layer has excitatory connections to itself and its immediate neighbours, 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.

2.9. Learning Algorithms

2.9.1. The Perceptron

This is a very simple model and consists of a single `trainable' neuron. Trainable means that its threshold and input weights are modifiable. Inputs are presented to the neuron and each input has a desired output (determined by us). If the neuron doesn't give the desired output, then it has made a mistake. To rectify this, its threshold and/or input weights must be changed. How this change is to be calculated is determined by the learning algorithm.

The output of the perceptron is constrained to Boolean values - (true, false), (1,0), (1, -1) or whatever. This is not a limitation because if the output of the perceptron were to be the input for something else, then the output edge could be made to have a weight. Then the output would be dependant on this weight.

The perceptron looks like

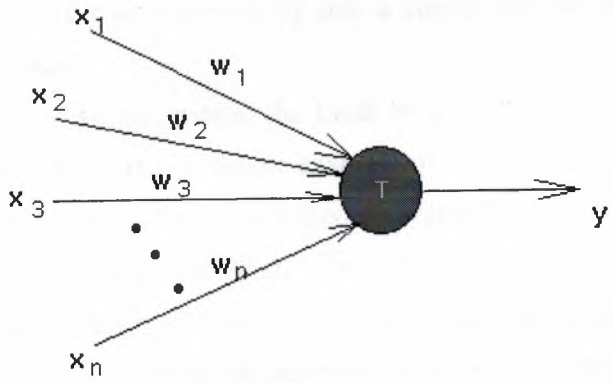

figure2.7. The perceptron.

x1, x2,..., xn are inputs. These could be real numbers or Boolean values depending on the problem.

y is the output and is Boolean.

w1, w2, ..., wn are weights of the edges and are real valued.

T is the threshold and is real valued.

The output y is 1 if the net input which is

$$w1\ x1 + w2\ x2 + \ldots + wn\ xn$$

Is greater than the threshold T. Otherwise the output is zero.

The idea is that we should be able to train this perceptron to respond to certain inputs with certain desired outputs. After the training period, it should be able to give reasonable outputs for any kind of input. If it wasn't trained for that input, then it should try to find the best possible output depending on how it was trained.
So during the training period we will present the perceptron with inputs one at a time and see what output it gives. If the output is wrong, we will tell it that it has made a mistake.

It should then change its weights and/or threshold properly to avoid making the same mistake later.

Note that the model of the perceptron normally given is slightly different from the one pictured here. Usually, the inputs are not directly fed to the trainable neuron but are modified by some "pre-processing units". These units could be arbitrarily complex, meaning that they could modify the inputs in any way. These units have been deliberately eliminated from our picture, because it would be helpful to know what can be achieved by just a single trainable neuron, without all its "powerful friends".

To understand the kinds of things that can be done using a perceptron, we shall see a rather simple example of its use - Compute the logical operations "and", "or", "not" of some given Boolean variables.

Computing "and": There are n inputs, each either a 0 or 1. To compute the logical "and" of these n inputs, the output should be 1 if and only if all the inputs are 1. This can easily be achieved by setting the threshold of the perceptron to n. The weights of all edges are 1. The net input can be n only if all the inputs are active.
Computing "or": It is also simple to see that if the threshold is set to 1, then the output will be 1 if at least one input is active. The perceptron in this case acts as the logical "or".

Computing "not": The logical "not" is a little tricky, but can be done. In this case, there is only one Boolean input. Let the weight of the edge be -1, so that the input, which is either 0 or 1, becomes 0 or -1. Set the threshold to 0. If the input is 0, the threshold is reached and the output is 1. If the input is -1, the threshold is not reached and the output is 0.

2.9.2. The XOR Problem

There are problems, which cannot be solved by any perceptron. In fact there are more such problems than problems, which can be solved using perceptrons. The most often quoted example is the XOR problem - build a perceptron, which takes 2 Boolean inputs and outputs the XOR of them. What we want is a perceptron, which will output 1 if the two inputs are different and 0, otherwise.

Input	Desired	Output
0	0	0
0	1	1
1	0	1
1	1	0

Consider the following perceptron as an attempt to solve the problem

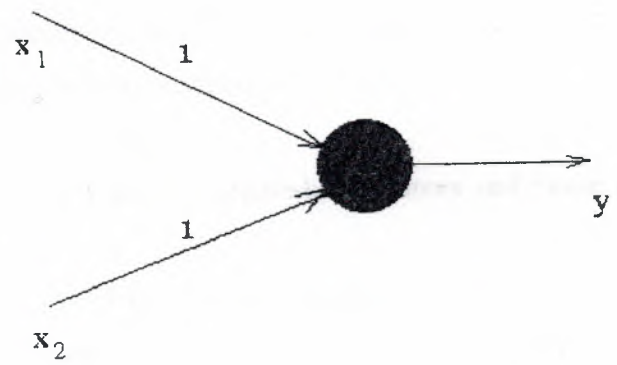

Figure 2.8. Example Illustrates The Perceptron Problem.

If the inputs are both 0, then net input is 0, which is less than the threshold (0.5). So the output is 0 - desired output.

If one of the inputs is 0 and the other is 1, then the net input is 1. This is above threshold, and so the output 1 is obtained.

But the given perceptron fails for the last case. To see that no perceptron can be built to solve the problem, try to build one yourself.

2.9.3. Pattern Recognition Terminology

The inputs that we have been referring to, of the form (x1, x2... xn) are also called as patterns. If a perceptron gives the correct, desired output for some pattern, then we say that the perceptron recognizes that pattern. We also say that the perceptron correctly classifies that pattern.

Since a pattern by our definition is just a sequence of numbers, it could represent anything such as a picture, a song, and a poem... anything that you can have in a computer file. We could then have a perceptron, which could learn such inputs and classify them, eg. A neat picture or a scribbling, a good or a bad song, etc. All we have to do is to present the perceptron with some examples -- give it some songs and tell it whether each one is good or bad. (It could then go all over the Internet, searching for songs, which you may like.) Sounds incredible? At least that's the way it is supposed to work. But it may not. The problem is that the set of patterns, which you want the perceptron to learn, might be something like the XOR problem. Then no perceptron can be made to recognize your taste.

2.9.4. Linearly Separable Patterns and Some Linear Algebra

If a set of patterns can be correctly classified by some perceptron, then such a set of patterns is said to be linearly separable. The term "linear" is used because the perceptron is a linear device. The net input is a linear function of the individual inputs and the output is a linear function of the net input. Linear means that there is no square (x2) or cube (x3), etc. terms in the formulas.

A pattern (x1,x2, ..., xn) is a point in an n-dimensional space. (Stop imagining things.) This is an extension of the idea that (x, y) is a point in 2-dimensions and (x, y, z) is a point in 3 dimensions. The utility of such a weird notion of an n-dimensional space is that there are many concepts, which are independent of dimension. Such concepts carry

Similarly, a straight line in 2D is given by -

$ax + by = c$

In 3D, a plane is given by -

$ax + by + cz = d$

When we generalize this, we get an object called as a hyper plane -

$w1x1 + w2x2 + ... + wnxn = T$

Notice something familiar? This is the net input to a perceptron. All points (patterns) for which the net input is greater than T belong to one class (they give the same output). All the other points belong to the other class.

We now have a lovely geometrical interpretation of the perceptron. A perceptron with weights w1, w2, ... wn and threshold T can be represented by the above hyper plane. All points on one side of the hyper plane belong to one class. The hyper plane (perceptron) divides the set of all points (patterns) into 2 classes.

Now we can see why the XOR problem cannot have a solution. Here there are 2 inputs. Hence there are 2 dimensions (luckily). The points that we want to classify are (0,0), (1,1) in one class and (0,1), (1,0) in the other class.

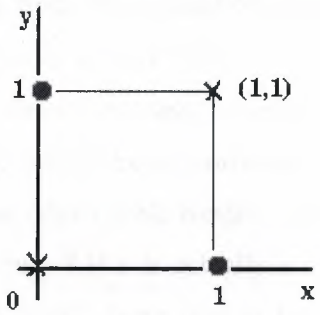

figure2.9. Two Inputs Dimensions.

Clearly we cannot classify the points (crosses on one side, circles on other) using a straight line. Hence no perceptron exists which can solve the XOR problem.

NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

INTELLECTUAL CONTROL SYSTEM FOR TECHNOLOGICAL PROCESSES

GRADUATION PROJECT
COM – 400

Student: Mohammed Alhaj Hussein (991288)

Supervisor: Assoc.Prof.Dr Rahib ABIYEV

Nicosia - 2002

ACKNOWLDGMENT

First of all I would like to thank Assoc. Prof. Dr. Rahib Abiyev for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

I would like to thank all of my friends who helped me to overcome my project especially Yousef, and manna.

Finally, I would like to thank my family, especially my parents. Their love and guidance saw me through doubtful times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together.

They have made countless sacrifices for my betterment. I can't repay them, but I do hope that their endless efforts will bear fruit and that I may lead them, myself and all who surround me to a better future.

ABSTRACT

Human beings epitomize the concept of "intelligent control." Despite its apparent computational advantage over humans, no machine or computer has come close to achieving the level of sensor-based control which humans are capable of. Thus, there is a clear need to develop computational methods which can abstract human decision-making processes based on sensory feedback.

Neural networks offer one such method with their ability to map complex nonlinear functions.

The aim of graduating project is the development of neural control system for technological processes. To achieve this aim the application problem of neural system for technological processes is considered .The model of neural systems, their architectures and learning algorithms are given.

Using neural structure the development of the neural control system is preformed, Controller is constructed on the base of neural network. The learning algorithm of neural network for controllers is described.

The modeling of the neural identification and control system is performed, Results of simulations of the developed and the traditional control system showed the improved time response characteristics of previous.

TABLE OF CONTENTS

INTRODUCTION

Researchers in the field of robotics and autonomous systems frequently find themselves looking towards human intelligence as a guide for developing "intelligent" machines. Paradoxically, control tasks, which seem easy or even trivial for humans, are often extremely difficult or impossible for computers or robots to duplicate. Rule-based systems usually fail to anticipate every eventuality and thus are ill suited for robots in uncertain and new environments. There is a clear need to develop computational methods which can, in a general framework, abstract the human decision-making process based on sensory feedback.

Modeling and identifying human control processes can be a significant step towards transferring human knowledge and skill in real-time control. This can lead to more intelligent control systems, and can bridge the gap between traditional artificial intelligence theory and the development of intelligent machines.

Artificial neural networks have shown great promise in identifying complex nonlinear systems. Thus, neural networks are well suited for generating the complex internal mapping from sensory inputs to control actions, which humans possess. Our goal is to develop a feasible neural network-based method for identifying human control strategy and transferring that control strategy to control systems. To this end, we are looking at an efficient and flexible neural network architecture that is capable of modeling nonlinear dynamic systems.

The project consists of introduction, 4 chapters and conclusion.

Chapter1 describes the states of neural control system however, its describes the two problems. First, the neural control of intelligent structures and the second, autonomous vehicle navigation.

Chapter2 describes the architecture of neural control systems for technological process, including the structure of neural system and descriptions of the functions of its main block are given. The neural network structures and their operation principles considering some problems, also the description of the learning in neural network has been considered, and some historical background of neural network has considered too.

Chapter3 describes the development of neural control system for technological process. The desired time response characteristic of system, neural control system's learning algorithm and characteristic of technological process are described. Using these the synthesis of procedures and simulation of neural control system are performed.

Chapter 4 describes the provide background information on this new architecture for neural network learning and a theoretical basis for its use. Then simulation results presented for this architecture in identifying both static and dynamic systems, including a nonlinear controller for an inverted pendulum system. Finally, some preliminary results in modeling human control strategy have been showed and discussed.

Conclusion presents the important obtained result that the project discussed and contributes in the project itself.

CHAPTER ONE. STATE OF APPLICATION PROBLEMS OF NEURAL NETWORK FOR TECHNOLOGICAL PROCESSES

1.1. Neural Control of Intelligent Structure

Smart structures as envisioned with embedded and distributed sensor/actor devices impose new challenges and problems for control engineering. The reasons for this are twofold: First, the design of embedded and distributed sensor/actor devices gives rise to new questions about the development of more appropriate control strategies interpreting the global versus local control strategy trade-off from a new and "distributed" perspective. Second, due to the non-linearity of the system components it is often too difficult to derive a system model of the smart structure suitable for classical controller design based on an exact analytical model using first principles.

Neural architectures such as neural networks offer in these cases the advantage to avoid the analytical modelling of smart structures and to "learn" the system transfer function from available experimental or simulated data instead. The work described here is focusing on the learning aspect of smart structure controllers with neural architectures and is organized along the following two main research directions of the basic research effort that aims at the development of novel neural control architectures. In this respect it has been aimed to resolve the "black-box-character" in neural network applications to allow a deeper mathematical analysis of the neural network after training.

This goal has been achieved through the introduction of the concept of dimensional homogeneity for neural networks [6], which leads to the emergence of dimensionless similarity parameters in the neural nets and allows to interpret the neural mapping in the network as the similarity function of the physical object under consideration, and, through the identification of the neural correspondence for classical control engineering techniques such as the Laplace-Transform [7]. It is expected that these two developments will ease a future performance analysis and a more direct comparison of classical controllers with neural control approaches including a future stability proof for neural control.

The practical development and design of novel controllers with neural architectures for different reference models [1,2,7]. These reference models are:

1) The tether deployment for small capsule re-entry,

2) The generic bump panel,

3) The adaptive helicopter blade,

4) The acoustic cavity.

In the practical development of neural controllers for these applications, the pre-processing of the training data, the employed training procedures for the neural network controller, the control performance and accuracy have been investigated. A generic procedure for the design of neural controllers has also been established for these purposes.

These two main research directions characterize the research results of the project A1 "Neural control of Intelligent Structures" which have been achieved in cooperation with other projects in the framework of the collaborative research project SFB 409 "Smart Structures in Aerospace Engineering". The details about the above mentioned different system models to be controlled and the simulation or experimental data have been provided from the partner project, while the neural modelling and the neural controller design has been performed in the project A1. The lessons learned and the results obtained are described in the following.

Two different neural network control schemes, a direct and an indirect control scheme, have been proposed in the literature [5]. For a detailed overview of neural control methodologies see [4]. While the direct neural control scheme in figure .1.1 doesn't use a model of the plant and is known to suffer from stability problems, the indirect neural control scheme makes use of a previously identified neural plant model, see figure1.2.

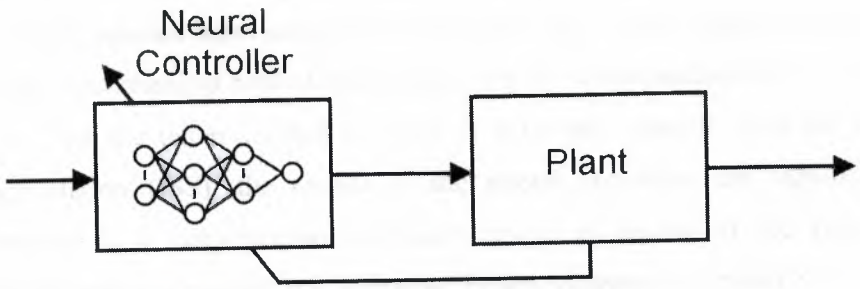

Figure .1.1. The direct neural control scheme

The neural plant model in figure 1.2 is trained using the squared error between the actual plant output and the model. Training is ceased when the approximation of the plant is good enough and the neural network can be used as a plant model for the

training of the controller without the use of the actual system. After successful plant identification, the neural controller is trained with an inverse training scheme [4] as shown in figure1.2. The control input is fed into the plant and the neural plant model.

The error between the commanded input and the plant output and the neural network output is then propagated back through the neural plant model using the first steps of the well-known Back propagation algorithm. The error found for the input neuron of the neural plant model corresponds directly to the error of the output neuron of the neural controller and can be used for the training with standard learning algorithms

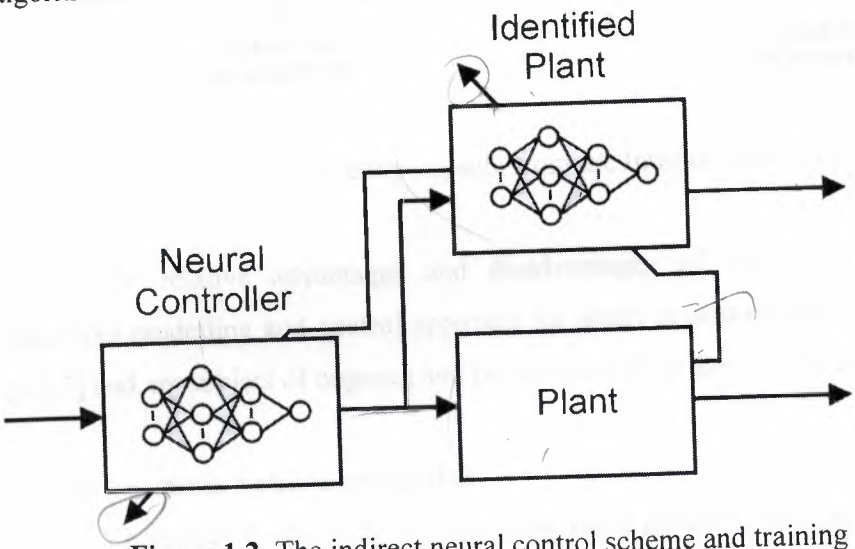

Figure 1.2. The indirect neural control scheme and training

The neural network controller is usually structured according to the neural network plant model using external feedback of the control signal and delayed values of the commanded input using time-delay lines. This neural control approach has been compared to classical controller designs using the mentioned reference examples.

For the tether-assisted de orbit of a re-entry capsule from the international space station (ISS), the results of the neural controller are significantly better compared to a conventional controller design as shown in the figure1.3.below, Together with the project B3 "Adaptive Tether Systems for Orbital Systems", a time-variant neural network controller has been developed for the deployment of a tethered re-entry capsule from the International Space Station (ISS).

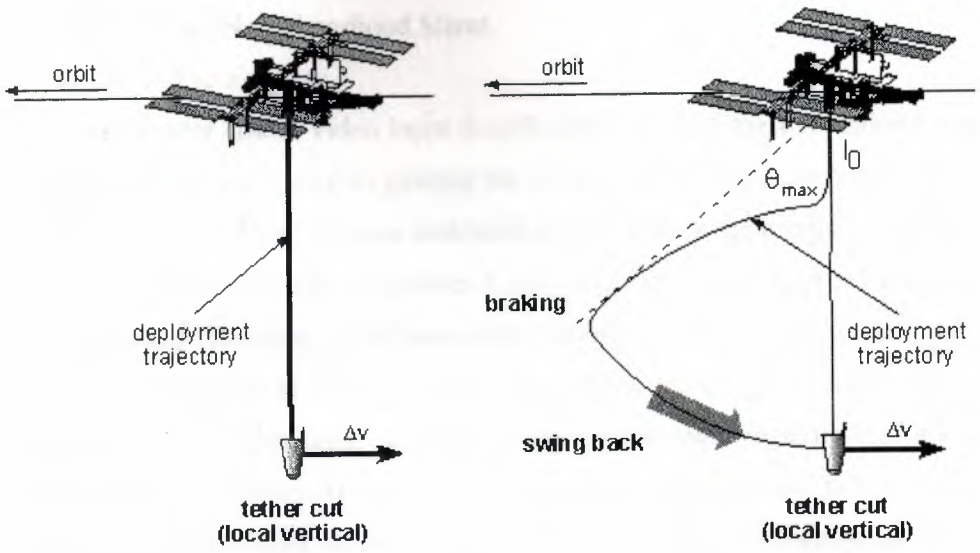

Figure 1.3. Tethered re-entry capsule from the International Space Station (ISS).

The relative advantages and disadvantages of the inductive versus the deductive modelling and control approach for smart structures have been reported in [1,2,7] and are subject of ongoing and future research in the SFB 409 project A1.

1.2. Autonomous Vehicle Navigation

Vision-based autonomous vehicle and robot guidance have proven difficult for algorithm-based computer vision methods, mainly because of the diversity of unexpected cases that must be explicitly dealt with in the algorithms and the real-time constraint, Pomerleau successfully demonstrated the potential of neural networks for overcoming these difficulties, his ALVINN (Autonomous Land Vehicle in Neural Networks) set a world record for autonomous navigation distance. After training on a two-mile stretch of highway, it drove the CMU Navlab. Equipped with video cameras and laser range sensors, for 21.2 miles with an average speed of 55 mph on a relatively old highway open to normal traffic, ALVINN was not disturbed by passing cars while it was being driven autonomously, ALVINN nearly doubled the previous distance world record for autonomous navigation. What is surprising is the simplicity of the networks and the training techniques used in ALVINN, which consists of several networks, each trained for a specific road situation:

1) Single-lane paved road.
2) Single-lane dirt road.

3) Two-lane Neighbourhood Street.

4) Multilane highway.

A monocular colour video input is sufficient for all of these situations; therefore, no depth perception is used in guiding the vehicle. Not using stereovision saves a significant amount of time, because matching of correspondence points in a stereo pair of images is computationally expensive. Laser rangefinder and laser reflectance inputs are also tested. The laser reflectance input resembles a black-and-white video image and can be handled in the same way as a video image. Reflectance input is advantageous over video input, because it appears the same regardless of the lighting conditions. This allows ALVINN to be trained in daylight and tested in darkness. Laser rangefinder input is useful for obstacle avoidance. However, a laser range image needs to be processed differently, because its image pixel values represent distance instead of lightness. We will focus the discussion on video image input.

A network in ALVINN for each situation consists of a single hidden layer of only four units, an output layer of 30 units and a 30 X 32 retina for the 960 possible input variables. The retina is fully connected to the hidden layer, and the hidden layer is fully connected to the output layer, as shown in figure1.4 for two representative nodes (out of a total of 960). The graph of the feed forward network is a node-coalesced cascade of directed versions of bipartite graphs K960, 4 and k4, 30 Pomerleau tried networks with more layers and more hidden units but did not observe significant performance improvement over this simple network, Because of the real-time constraint of the task, a simple network is definitely preferred.

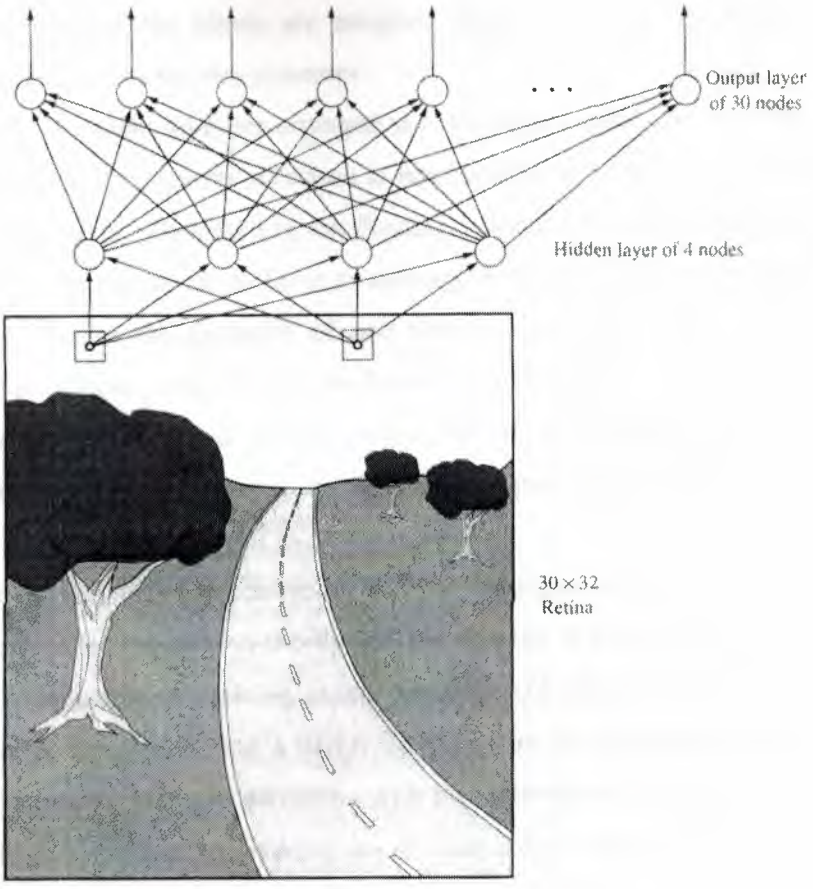

Figure 1.4. The graph of a network in ALVINN.

It is a node-coalesced of two directed bipartite graphs, The Image on the retina is a low-resolution version of a cooler video image with 480 X 512 pixels. A 16X16 neighbourhood in the video image is randomly sampled and averaged to produce a single pixel on the retina .the outputs from the three channels of a colour video image-namely, red(R), green (G), and blue (B) are combined to produce

$$P = \frac{B}{255} + \frac{B}{R+G+B}$$

Where, P is the brightness of the combined image. This combination is based on empirical observation. What is interesting is that it approximates the learning result if one chooses to add another layer to learn the pre-processing from video image to the retina. The darkest 5% of the pixels on the retina are assigned the minimum activation level of -1, and the brightest 5% are assigned the maximum activation level of 1. The

remaining 90% of the pixels are assigned activation values proportional to their brightness relative to the two extremes.

The 30 output units are arranged in a one-dimensional array for controlling the steering wheel. The steering direction is represented by a Gaussian activation pattern in the output layer, illustrated in the Figure1.5; the distributed pattern representation of the output proves to be useful in evaluating the reliability of the network output. If the vehicle under the guidance of one network (e.g., for single-lane paved road) transits into a new situation (e.g. multilane highway) the network will be confused. There is a high likelihood that the output pattern will significantly deviate from a Gaussian pattern. This signals; the ALVINN to pick another network to guide the vehicle.

Each network is trained using the back propagation algorithm with a technique Pomerleau called training-on-the-fly: i.e. the network is trained by observing a person driving a sequence of training pairs, consisting of input images and the person's response, is obtained during a drive. Training can be performed at the same time. There are several potential problems with the training-on-the-fly approach. They are all due to the low level of diversity or- in other words, the high level of similarity, in the training data. For example, the network needs to learn how to recover from Various mistakes, a sequence of consistently similar training data will also cause the network to over learn the current situation and forget about what it might have learned about other situations, Diversity in the training data is necessary for valid generalization. Pomerleau used several techniques to solve these problems.

First, the inner images and the steering directions are geometrically transformed as if the vehicle had been in different positions relative to the road. Second, structured noise is added to the input images to simulate different situations on the road, such as passing cars. Guardrails, and trees. New training pairs are formed using a new image with added structure noise and the same steering direction as the noise-free image. These techniques greatly increase the diversity of the training data. Thus leading to good generalisation of learning.

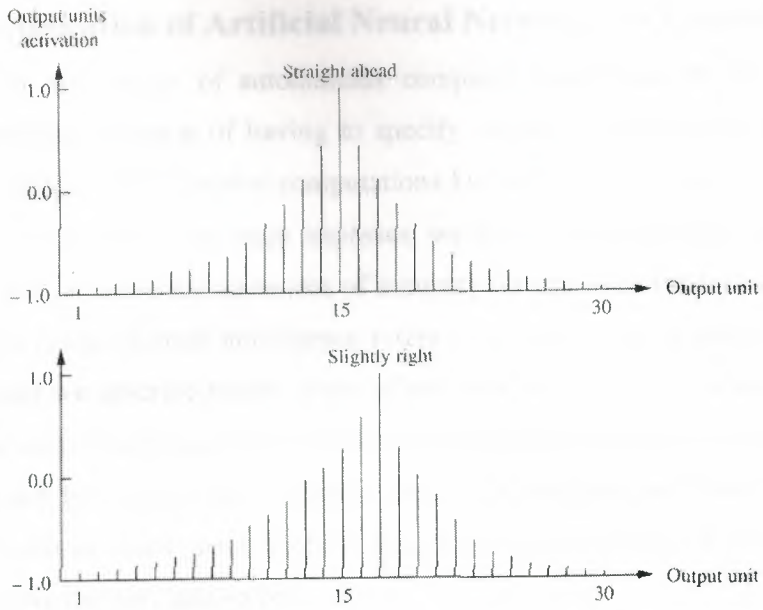

figure1.5. Representation of two steering directions in the output layer from the plot of output unit activation values versus the output unit.

After each network is trained for a specific situation, how is the system going to decide when to use which network? Pomerleau proposed two techniques for selecting a network: output appearance and input reconstruction. Output appearance measures the deviation of the shape of the output response from the Gaussian shape. Although not always true, there is a high level of correlation between the Gaussian shape of the output response and the applicability of the network to the current situation. Input reconstruction feeds the output response back through the connections and the hidden layer to reconstruct an input image. The difference between the real input image and the reconstructed image provides another indication of the applicability of the network to the current situation. These two techniques can then be used to guide the choice of the right network for the current situation, Due to the Fact that neural networks are not good at remembering maps and planning the route, a symbolic component is added to the system for these functions. The symbolic component is also responsible for generating structured noises and transformations to increase the diversity of the training data and for coordinating all the components in the system.

1.3. Application of Artificial Neural Network 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 suspect of being so. To circumvent such impasses, we resort to complexity reducing strategies and tactics, which trade some loss of accuracy for significant reduction in complexity; the term computational intelligence refers to such complexity reduction methods. In this paper we describe briefly some of our own work in this area and then develop a computation intelligence view of the tasks of process monitoring and optimization, as performed by autonomous system. Some important current fields of discovery in computational intelligence include neural net computing evolutionary, fuzzy sets, associative memory and so on.

Some of the theory bounded evolutional trends in real time al application are pointed out based system. The evolutional main stream is increasing interdisciplinary integration.

Three sub trends are illustrated on examples: mechanical combination of method, 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 system designed 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. The most attention is given to the neural based force adaptation level implemented by three-layered back propagation neural network. A computer based simulation system for the big_in_hole insertion task is developed to analyse the capabilities of the neural controllers. Their behaviour 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 of the UK has supported the transfer of neural technology as part of the campaign, the university of the new castle and EDS advanced technologies group have set-up a process monitoring and control club.

The logical design of a neural controller is a achieved by representing a neural computation as a stochastic timed linear proof with a built-in system for rewards and punishments based on the timelines of a computation performed by a neural controller.

Logical designs are represented with stochastic forms of proof nets and proof boxes. Sample application of the logical design methodology of the truck-backer upper and a real time object recognition and tracking system (RTorts) are presented. Performance result of the implementation module of the (RTorts) are given and compared to similar system.

The work describe in the neural network of intelligent structure is focusing on the learning aspect of smart structure controllers with neural architectures along the following two main research directions of The basic research effort that aims at the development of novel neural control architectures. So it used two different neural network control schemes, a direct and an indirect control scheme.

For the autonomous navigation concept is explaining a network in ALVININ for each situation consists of a single hidden layer of only four units. The retina is fully connected to the hidden layer as well as hidden layer is fully connected to the output layer as a result of the real time constraint of the task, a simple network is definitely preferred.

Each network is trained using the back propagation algorithm with a technique Pomerleau called training-on-the-fly; training can be performed at the same time.

Pomerleau proposed two techniques for selecting a network: output appearance and input reconstruction as we have declared before, this two techniques can then be used to guide the choice of the right network for the current situation.

CHAPTER TWO. STRUCTURE AND LEARNING OF NEURAL NETWORKS

2.1. Introduction To Neural Networks

The power and speed of modern digital computers is truly astounding. No human can ever hope to compute a million operations a second. However, there are some tasks for which even the most powerful computers cannot compete with the human brain, perhaps not even with the intelligence of an earthworm.

Imagine the power of the machine, which has the abilities of both computers and humans. It would be the most remarkable thing ever. And all humans can live happily ever after. This is the aim of artificial intelligence in general.

When we are talking about a neural network, we should more properly say "artificial neural network" (ANN), because that is what we mean most of the time. Biological neural networks are much more complicated than the mathematical models we use for ANNs. But it is customary to be lazy and drop the "A" or the "artificial".

An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information.

The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

At the core of neural computation are the concepts of distributed, adaptive, and non-linear computing. Neural networks perform computation in a very different way than conventional computers, where a single central processing unit sequentially dictates every piece of the action.

Evolving from neuro-biological insights, neural network technology gives a computer system an amazing capacity to actually learn from input data. Artificial neural networks have provided solutions to problems normally requiring human observation and thought processes. Some real world applications include:

- Quality Control
- Financial Forecasting
- Economic Forecasting
- Credit Rating
- Speech & Pattern Recognition
- Biomedical Instrumentation
- Process Modelling & Management
- Laboratory Research
- Oil & Gas Exploration
- Health Care Cost Reduction
- Targeted Marketing
- Defence Contracting
- Bankruptcy Prediction
- Machine Diagnostics
- Securities Trading

2.2 Some Other Definitions of a Neural Networks

According to the DARPA Neural Network Study (1988, AFCEA International Press, p. 60): a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

According to Haykin, S. (1994), Neural Networks: A Comprehensive Foundation, NY: Macmillan, p. 2:

A neural network is a massively parallel-distributed processor that has a natural propensity for storing experiential knowledge and making it available for use, It resembles the brain in two respects:

1.The network through a learning process acquires knowledge.

2.Interneuron connection strengths known as synaptic weights are used to store the knowledge.

ANNs have been applied to an increasing number of real-world problems of considerable complexity. Their most important advantage is in solving problems that are too complex for conventional technologies problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found.

ANNs have been applied to an increasing number of real-world problems of considerable complexity. Their most important advantage is in solving problems that are too complex for conventional technologies problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found. In general, because of their abstraction from the biological brain, ANNs are well suited to problems that people are good at solving, but for which computers are not. This problem includes pattern recognition and forecasting (which requires the recognition of trends in data).

Neural Networks approaches this problem by trying to mimic the structure and function of our nervous system. Many researchers believe that AI (Artificial Intelligence) and neural networks are completely opposite in their approach. Conventional AI is based on the symbol system hypothesis. Loosely speaking, a symbol system consists of indivisible entities called symbols, which can form more complex entities, by simple rules. The hypothesis then states that such a system is capable of and is necessary for intelligence.

The general belief is that Neural Networks is a sub-symbolic science. Before symbols themselves are recognized, some thing must be done so that conventional AI can then manipulate those symbols. To make this point clear, consider symbols such as cow, grass, house etc. Once these symbols and the "simple rules" which govern them are known, conventional AI can perform miracles. But to discover that something is a cow is not trivial. It can perhaps be done using conventional AI and symbols such as - white, legs, etc. But it would be tedious and certainly, when you see a cow, you instantly recognize it to be so, without counting its legs.

But this belief that AI and Neural Networks are completely opposite is not valid because, even when you recognize a cow, it is because of certain properties, which you observe, that you conclude that it is a cow. This happens instantly because various parts of the brain function in parallel. All the properties, which you observe, are "summed up". Certainly there are symbols here and rules - "summing up". The only difference is that in AI, symbols are strictly indivisible, whereas here, the symbols (properties) may occur with varying degrees or intensities.

Only breaking this line of distinction between AI and Neural Networks, and combining the results obtained in both, towards a unified framework, can make progress in this area.

2.3. Biological Information Process

To imitate biological information processing models for different level 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 behaviour of the neuron. On the network level The interconnection of identical neurons to form network is examined to describe specific sensor and motor city-related functions such as filtering, projection operations, controller function. In non-linear, biological system, network 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.

2.3.1 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 non-linear 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.

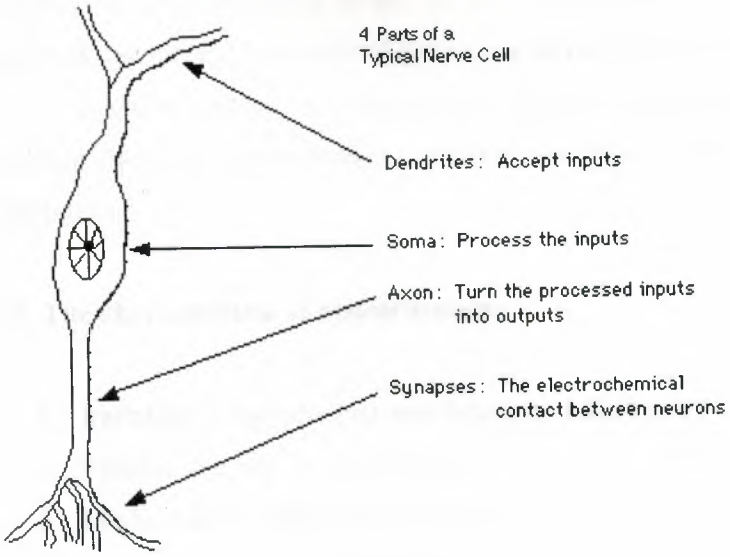

figure2.1. Biological Neuron

2.3.2 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.

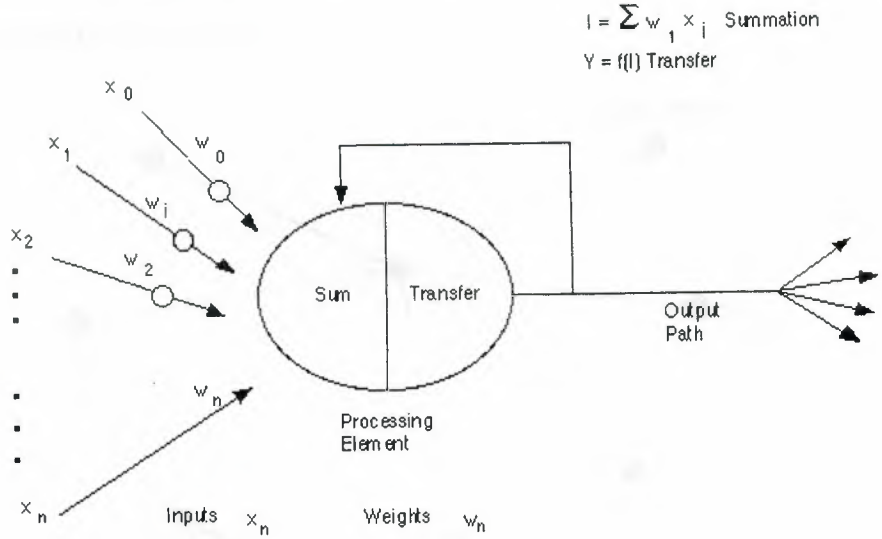

Figure2.2. Artificial Neuron

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.

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.

2.4. The characteristic of neural systems

1. Imitation of the structure and function of the brain.
2. Parallel information processing.
3. Implicit knowledge representation.
4. Application of inductive reasoning.
5. Learning occurs within the system.

2.5. The Structure of the Nervous System

For our purpose, it will be sufficient to know that the nervous system consists of neurons, which are connected to each other in a rather complex way. Each neuron can be thought of as a node and the interconnections between them are edges as shown below in the figure2.3:

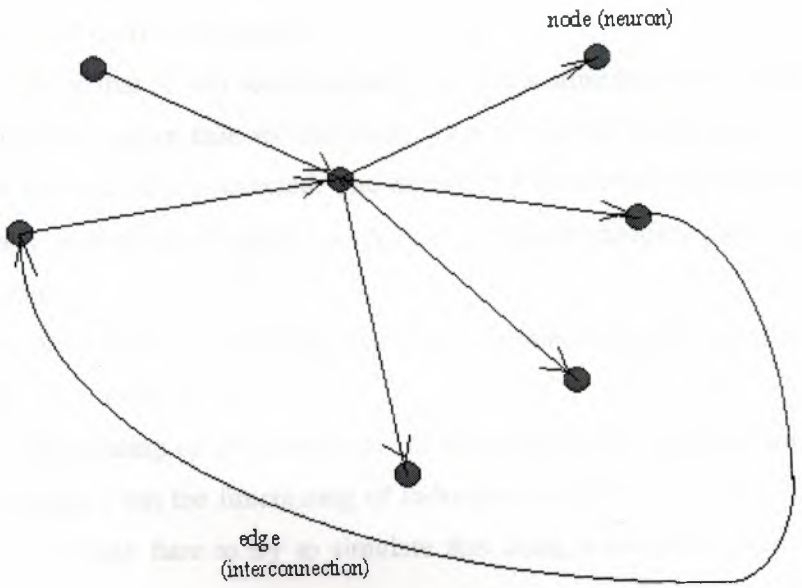

figure2.3. Edge interconnections.

Such a structure is called as a directed graph. Further, each edge has a weight associated with it, which represents how much the two neurons, which are connected by it, can interact. If the weight is more, then the two neurons can interact much more a stronger signal can pass through the edge.

2.6. Functioning of the Nervous System

The nature of interconnections between 2 neurons can be such that one neuron can either stimulate or inhibit the other. An interaction can take place only if there is an edge between 2 neurons. If neuron A is connected to neuron B as below with a weight w, in the figure1.2

Figure2.4. The Edge Between Two Neurons.

Then if A is stimulated sufficiently, it sends a signal to B. The signal depends on the weight w, and the nature of the signal, whether it is stimulating or inhibiting. This depends on whether w is positive or negative. If sufficiently strong signals are sent, B may become stimulated.

Note that A will send a signal only if it is stimulated sufficiently, that is, if its stimulation is more than its threshold. Also if it sends a signal, it will send it to all nodes to which it is connected. The threshold for different neurons may be different. If many neurons send signals to A, the combined stimulus may be more than the threshold.

Next if B is stimulated sufficiently, it may trigger a signal to all neurons to which it is connected.

Depending on the complexity of the structure, the overall functioning may be very complex but the functioning of individual neurons is as simple as this. Because of this we may dare to try to simulate this using software or even special purpose hardware.

2.7 The Difficulty of Modelling a Brain-like Neural Network

We have seen that the functioning of individual neurons is quite simple. Then why is it difficult to achieve our goal of combining the abilities of computers and humans?

The difficulty arises because of the following:

It is difficult to find out which neurons should be connected to which. This is the problem of determining the neural network structure. Further, the interconnections in the brain are constantly changing. The initial interconnections seem to be largely governed by genetic factors.

The weights on the edges and thresholds in the nodes are constantly changing. This problem has been the subject of much research and has been solved to a large extent. The approach has been as follows: Given some input, if the neural network makes an error, then it can be determined exactly which neurons were active before the error. Then we can change the weights and thresholds appropriately to reduce this error.

For this approach to work, the neural network must "know" that it has made a mistake. In real life, the mistake usually becomes obvious only after a long time. This situation is more difficult to handle since we may not know which input led to the error.

Also notice that this problem can be considered as a generalization of the previous

problem of determining the neural network structure. If this is solved, that is also solved. This is because if the weight between two neurons is zero then, it is as good as the two neurons not being connected at all. So if we can figure out the weights properly, then the structure becomes known. But there may be better methods of determining the structure.

The functioning of individual neurons may not be so simple after all. For example, remember that if a neuron receives signals from many neighbouring neurons, the combined stimulus may exceed its threshold. Actually, the neuron need not receive all signals at exactly the same time, but must receive them all in a short time-interval.

It is usually assumed that such details will not affect the functioning of the simulated neural network much. But may be it will.

Another example of deviation from normal functioning is that some edges can transmit signals in both directions. Actually, all edges can transmit in both directions, but usually they transmit in only 1 direction, from one neuron to another.

2.8.Neural Network Topologies

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

Neural networks are built from a large number of very simple processing elements that individually deal with pieces of a big problem. A processing element (PE) simply multiplies an input by a set of weights, and a nonlinearly transforms the result into an output value. The principles of computation at the PE level are deceptively simple. The power of neural computation comes from the massive interconnection among the PEs, which share the load of the overall processing task, and from the adaptive nature of the parameters (weights) that interconnect the PEs.

Normally, a neural network will have several layers of PEs. The most basic and commonly used neural network architecture is the multi layer perceptron (MLP). The diagram (figure 2.5.) below illustrates a simple MLP. The circles are the PEs arranged in layers. The left row is the input layer, the middle row is the hidden layer, and the right row is the output layer. The lines represent weighted connections (i.e., a scaling factor) between PEs.

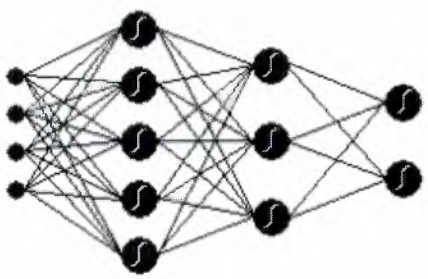

figure2.5. A simple Multi Layer Perceptron

The performance of an MLP is measured in terms of a desired signal and an error criterion. The output of the network is compared with a desired response to

produce an error. An algorithm called back propagation is used to adjust the weights a small amount at a time in a way that reduces the error. The network is trained by repeating this process many times. The goal of the training is to reach an optimal solution based on the performance measurement.

We shall now try to understand different types of neural networks

2.8.1. 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.

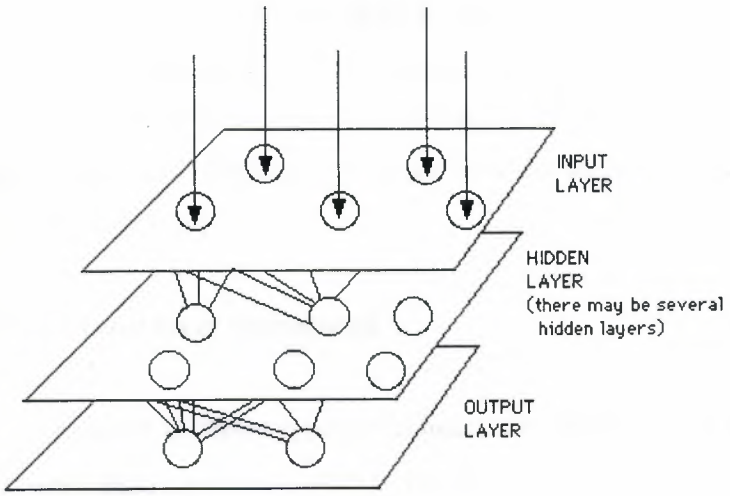

figure2.6. Layer Structure

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.

2.8.2. 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

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.

2.8.2.1 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.

2.8.2.2 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-centre/off surround** A neuron within a layer has excitatory connections to itself and its immediate neighbours, 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.

2.9. Learning Algorithms

2.9.1. The Perceptron

This is a very simple model and consists of a single `trainable' neuron. Trainable means that its threshold and input weights are modifiable. Inputs are presented to the neuron and each input has a desired output (determined by us). If the neuron doesn't give the desired output, then it has made a mistake. To rectify this, its threshold and/or input weights must be changed. How this change is to be calculated is determined by the learning algorithm.

The output of the perceptron is constrained to Boolean values - (true, false), (1,0), (1, -1) or whatever. This is not a limitation because if the output of the perceptron were to be the input for something else, then the output edge could be made to have a weight. Then the output would be dependant on this weight.

The perceptron looks like

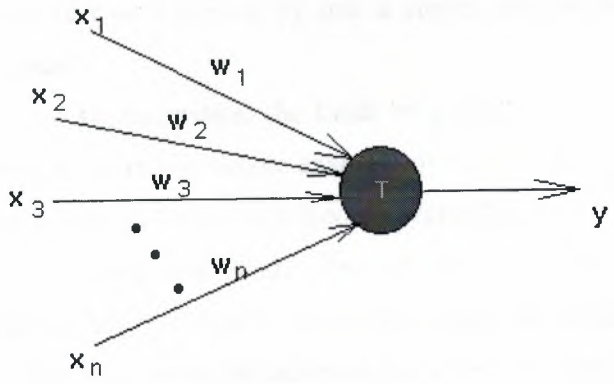

figure2.7. The perceptron.

x1, x2,..., xn are inputs. These could be real numbers or Boolean values depending on the problem.

y is the output and is Boolean.

w1, w2, ..., wn are weights of the edges and are real valued.

T is the threshold and is real valued.

The output y is 1 if the net input which is

$$w1\ x1 + w2\ x2 + \ldots + wn\ xn$$

Is greater than the threshold T. Otherwise the output is zero.

The idea is that we should be able to train this perceptron to respond to certain inputs with certain desired outputs. After the training period, it should be able to give reasonable outputs for any kind of input. If it wasn't trained for that input, then it should try to find the best possible output depending on how it was trained.
So during the training period we will present the perceptron with inputs one at a time and see what output it gives. If the output is wrong, we will tell it that it has made a mistake.

It should then change its weights and/or threshold properly to avoid making the same mistake later.

Note that the model of the perceptron normally given is slightly different from the one pictured here. Usually, the inputs are not directly fed to the trainable neuron but are modified by some "pre-processing units". These units could be arbitrarily complex, meaning that they could modify the inputs in any way. These units have been deliberately eliminated from our picture, because it would be helpful to know what can be achieved by just a single trainable neuron, without all its "powerful friends".

To understand the kinds of things that can be done using a perceptron, we shall see a rather simple example of its use - Compute the logical operations "and", "or", "not" of some given Boolean variables.

Computing "and": There are n inputs, each either a 0 or 1. To compute the logical "and" of these n inputs, the output should be 1 if and only if all the inputs are 1. This can easily be achieved by setting the threshold of the perceptron to n. The weights of all edges are 1. The net input can be n only if all the inputs are active.
Computing "or": It is also simple to see that if the threshold is set to 1, then the output will be 1 if at least one input is active. The perceptron in this case acts as the logical "or".

Computing "not": The logical "not" is a little tricky, but can be done. In this case, there is only one Boolean input. Let the weight of the edge be -1, so that the input, which is either 0 or 1, becomes 0 or -1. Set the threshold to 0. If the input is 0, the threshold is reached and the output is 1. If the input is -1, the threshold is not reached and the output is 0.

2.9.2. The XOR Problem

There are problems, which cannot be solved by any perceptron. In fact there are more such problems than problems, which can be solved using perceptrons. The most often quoted example is the XOR problem - build a perceptron, which takes 2 Boolean inputs and outputs the XOR of them. What we want is a perceptron, which will output 1 if the two inputs are different and 0, otherwise.

Input	Desired	Output
0	0	0
0	1	1
1	0	1
1	1	0

Consider the following perceptron as an attempt to solve the problem

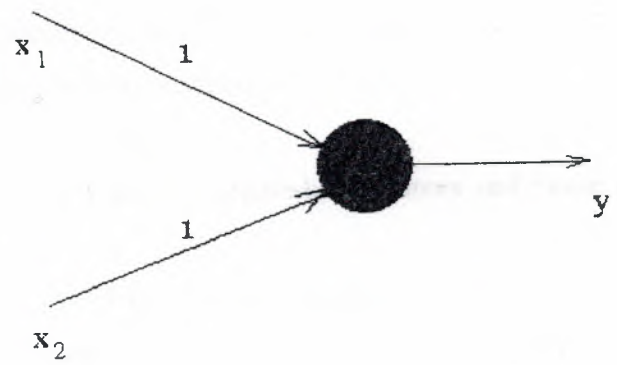

Figure 2.8. Example Illustrates The Perceptron Problem.

If the inputs are both 0, then net input is 0, which is less than the threshold (0.5). So the output is 0 - desired output.

If one of the inputs is 0 and the other is 1, then the net input is 1. This is above threshold, and so the output 1 is obtained.

But the given perceptron fails for the last case. To see that no perceptron can be built to solve the problem, try to build one yourself.

2.9.3. Pattern Recognition Terminology

The inputs that we have been referring to, of the form (x1, x2... xn) are also called as patterns. If a perceptron gives the correct, desired output for some pattern, then we say that the perceptron recognizes that pattern. We also say that the perceptron correctly classifies that pattern.

Since a pattern by our definition is just a sequence of numbers, it could represent anything such as a picture, a song, and a poem... anything that you can have in a computer file. We could then have a perceptron, which could learn such inputs and classify them, eg. A neat picture or a scribbling, a good or a bad song, etc. All we have to do is to present the perceptron with some examples -- give it some songs and tell it whether each one is good or bad. (It could then go all over the Internet, searching for songs, which you may like.) Sounds incredible? At least that's the way it is supposed to work. But it may not. The problem is that the set of patterns, which you want the perceptron to learn, might be something like the XOR problem. Then no perceptron can be made to recognize your taste.

2.9.4. Linearly Separable Patterns and Some Linear Algebra

If a set of patterns can be correctly classified by some perceptron, then such a set of patterns is said to be linearly separable. The term "linear" is used because the perceptron is a linear device. The net input is a linear function of the individual inputs and the output is a linear function of the net input. Linear means that there is no square (x2) or cube (x3), etc. terms in the formulas.

A pattern (x1,x2, ..., xn) is a point in an n-dimensional space. (Stop imagining things.) This is an extension of the idea that (x, y) is a point in 2-dimensions and (x, y, z) is a point in 3 dimensions. The utility of such a weird notion of an n-dimensional space is that there are many concepts, which are independent of dimension. Such concepts carry

Similarly, a straight line in 2D is given by -

$ax + by = c$

In 3D, a plane is given by -

$ax + by + cz = d$

When we generalize this, we get an object called as a hyper plane -

$w1x1 + w2x2 + ... + wnxn = T$

Notice something familiar? This is the net input to a perceptron. All points (patterns) for which the net input is greater than T belong to one class (they give the same output). All the other points belong to the other class.

We now have a lovely geometrical interpretation of the perceptron. A perceptron with weights w1, w2, ... wn and threshold T can be represented by the above hyper plane. All points on one side of the hyper plane belong to one class. The hyper plane (perceptron) divides the set of all points (patterns) into 2 classes.

Now we can see why the XOR problem cannot have a solution. Here there are 2 inputs. Hence there are 2 dimensions (luckily). The points that we want to classify are (0,0), (1,1) in one class and (0,1), (1,0) in the other class.

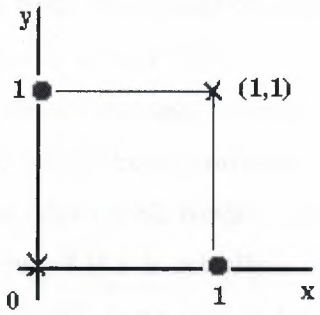

figure2.9. Two Inputs Dimensions.

Clearly we cannot classify the points (crosses on one side, circles on other) using a straight line. Hence no perceptron exists which can solve the XOR problem.

2.9.5 Perceptron Learning Algorithms

During the training period, a series of inputs are presented to the perceptron - each of the form (x1, x2... xn). For each such input, there is a desired output - either 0 or 1. ` The net input, which is w1 x1, determines the actual output + w2 x2 + ... + wn xn. If the net input is less than threshold then the output is 0, otherwise output is 1. If the perceptron gives a wrong (undesirable) output, then one of two things could have happened -

1. The desired output is 0, but the net input is above threshold. So the actual output becomes in such a case we should decrease the weights. But by how much? The perceptron-learning algorithm says that the decrease in weight of an edge should be directly proportional to the input through that edge. So, New weight of an edge i = old weight – cxi There are several algorithms depending on what c is. For now, think that it is a constant.

The idea here is that if the input through some edge was very high, then that edge must have contributed to most of the error. So we reduce the weight of that edge more (i.e. proportional to the input along that edge).

2.The other case when the perceptron makes a mistake is when the desired output is 1, but the net input is below threshold.

Now we should increase the weights. Using the same intuition, the increase in weight of an edge should be proportional to the input through that edge. So,

New weight of an edge i = old weight + cxi

What about c? If c is actually a constant, then the algorithm is called as the "fixed increment rule". Note that in this case, the perceptron may not correct its mistake immediately. That is, when we change the weights because of a mistake, the new weights don't guarantee that the same mistake will not be repeated. This could happen if c is very small. However, by repeated application of the same input, the weights will change slowly each time, until that mistake is avoided.

We could also choose c in such a way that it will certainly avoid the most recent mistake, next time it is presented the same input. This is called as the "absolute correction rule". The problem with this approach is that by learning one input, it might "forget" a previously learnt input. For example, if one input leads to an increase in some weight and another input decreases it, then such a problem may arise.

2.10.Neural network Learning

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

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

The training method usually consists of one of three schemes:

2.10.1.Unsupervised learning

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

2.10.2.Reinforcement learning

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

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

2.10.3. Error Back propagation

This method is proven highly successful in training of multi-layered neural nets, The network is not just given reinforcement for how it is doing on a task. Information about errors is also filtered back through the system and is used to adjust the connections between the layers, thus improving performance. A form of supervised learning.

However, when we have a multi-layer network we encounter a difficulty: we don't have any target values for the hidden units. 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 favour after an initial period of high popularity in the 1950s .it took 30 years before the error back propagation algorithm popularised away to train hidden units, leading to a new wave of a neural network research and applications.

For hidden units, we must propagate the error back from the outputs nodes. Again using the chain rule, we can expand the error of a hidden unit in terms of its posterior nodes:

$$\delta_j = -\sum_{i \in p_j} \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial y_i} \frac{\partial y_j}{\partial net_j} \tag{2.1}$$

Of the three factor inside the sums, the first is just the error of node i. The second is:

$$\frac{\partial net_i}{\partial y_i} = \frac{\partial}{\partial y_i} \sum_{k \in AJ} w_{ik} y_k = w_{ij} \tag{2.2}$$

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

$$\frac{\partial y_i}{\partial net_j} = \frac{\partial f_i(net_j)}{\partial net_j} = f_j'(net_j) \tag{2.3}$$

For a 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:

$$f_h'(net_h) = 1 - y_h^2 \tag{2.4}$$

Putting all the pieces together we get

$$\partial_j = f_j'(net_j) \sum_{i \in pj} \delta_i w_{ij} \tag{2.5}$$

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.

2.10.4. Learning laws

There are a variety of learning laws, which are in common use. These laws are mathematical algorithms used to update the connection weights. Most of these laws are some sorts of variation of the best-known and oldest learning law, Hebb's Rule. Man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplification represented by the learning laws currently developed. Research into different learning functions continues as new ideas routinely show up in trade publications etc. A few of the major laws are given as an example below.

2.10.4.1.Hebb's Rule

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

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

2.10.4.2.Hopfield Law

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

2.10.4.3. The Delta Rule

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

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

2.10.4.4.Kohonen's Learning Law

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

The Kohonen rule does not require desired output. Therefore it is implemented in the unsupervised methods of learning. Kohonen has used this rule combined with

the on-centre/off-surround intra- layer connection (discussed earlier) to create the self-organizing neural network, which has an unsupervised learning method.

2.11. Recurrent Network

Consider the following two networks

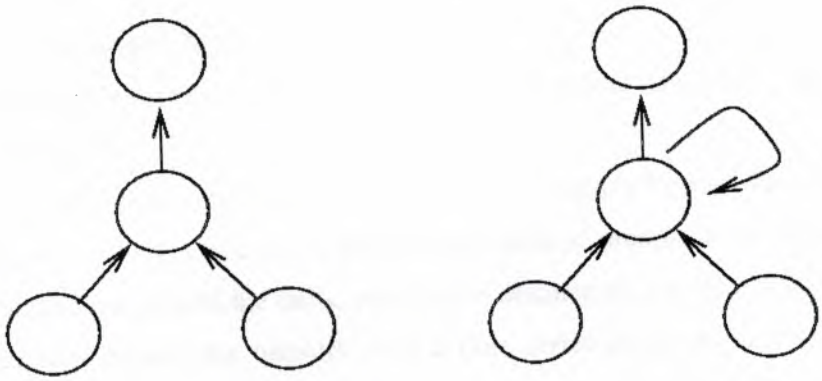

Figure 2.10. Feed forward networks

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 behaviour is based on its history, and so we must think of pattern presentation as it happens in time.

2.11.1. 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:

$$y_i(t) = f_i(net_i(t-1)) \qquad (2.6)$$

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. Its 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 .

2.11.2. 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

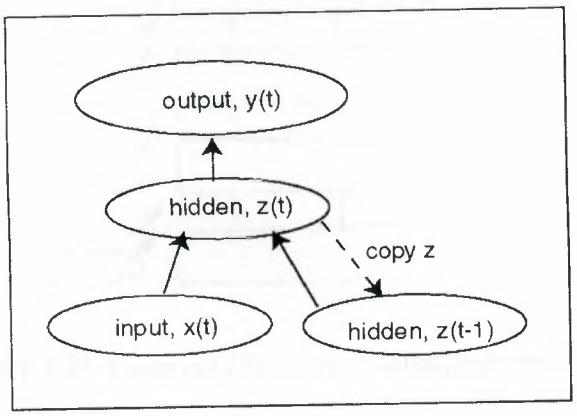

Figure 2.11. Simple recurrent neural network

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 back propagation 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 time steps. 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 also all inputs seen so far by the network. The figure below illustrates one version of the process:

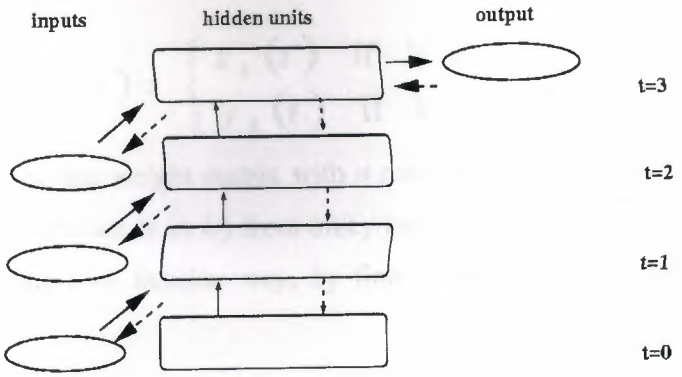

Figure 2.12. Learning of recurrent neural networks

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 modelling). The main task is 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.

2.11.3. Real Time Recurrent Learning

In deriving a gradient-based update rule for recurrent networks, we now make network connectivity 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

$$z_k(t) = \begin{cases} x_k(t) & \text{if } k \in I \\ y_k(t) & \text{if } k \in U \end{cases} \qquad (2.7)$$

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:

$$net_k(t) = \sum_{l \in U \cup I} w_{kl} z_l(t) \qquad (2.8)$$

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

$$y_k(t+1) = f_k(\text{net}_k(t)) \qquad (2.9)$$

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

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases} \qquad (2.10)$$

and define our error function for a single time step as

$$E(\tau) = \frac{1}{2} \sum_{k \in U} \left[e_k(\tau)^2 \right] \qquad (2.11)$$

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

$$E_{total}(t_0, t) = \sum_{\tau = t_{0+1}}^{t_1} E(\tau) \qquad (2.12)$$

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:

$$\nabla_W E_{total}(t_0, t+1) = \nabla_w E_{total}(t_0, t) + \nabla_w E(t+1) \qquad (2.13)$$

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

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}} \qquad (2.14)$$

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

$$\sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t) \qquad (2.15)$$

We therefore need an algorithm that computes

$$\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in U} \frac{\partial E(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \qquad (2.16)$$

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.

$$\partial y_k(t) \Big/ \partial w_{ij} \qquad (2.17)$$

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

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k'(net_k(t)) \left[\sum_{l \in U \cup I} w_{kl} \frac{\partial z_l(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right] \qquad (2.18)$$

Where δ_{ik} is the Kronecker delta

$$\delta_{ik} = \begin{cases} 1 & \text{if i = k} \\ 0 & \text{otherwise} \end{cases} \qquad (2.19)$$

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

$$\frac{\partial z_l(t)}{\partial w_{ij}} = 0 \text{ for } l \in I \qquad (2.20)$$

Equation becomes:

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k'(net_k(t))\left[\sum_{I \in U} w_{kI} \frac{\partial y_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t)\right] \qquad (2.21)$$

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

$$\frac{\partial y_k(t_0)}{\partial w_{ij}} = 0 \qquad (2.22)$$

These equations hold for all

$$k \in U, i \in U \text{ and } j \in U \cup I \qquad (2.23)$$

We therefore need to define the values

$$P_{ij}^k(t) = \frac{\partial y_k(t)}{\partial w_{ij}} \qquad (2.24)$$

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

$$P_{ij}^I(t+1) = f_k'(net_k(t))\left[\sum_{I \in U} w_{kI} P_{ij}^I + \delta_{ik} z_j(t)\right] \qquad (2.24)$$

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

$$\Delta w_{ij}(t) = \mu \sum_{k \in U} e_k(t) P_{ij}^k(t) \qquad (2.25)$$

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

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t) \qquad (2.26)$$

2.12. Advantages of the neural network

Either humans or other computer techniques can use neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, to extract patterns and detect trends that are too complex to be noticed. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

2.13. Neural network in practice

Given this description of neural networks and how they work, what real world applications are they suited for? Neural networks have broad applicability to real world business problems. In fact, they have already been successfully applied in many industries.

Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs including Sales forecasting :

- Industrial process control
- Customer research
- Data validation
- Risk management
- Target marketing

But to give you some more specific examples; ANN are also used in the following specific paradigms: recognition of speakers in communications; diagnosis of hepatitis;

recovery of telecommunications from faulty software; interpretation of multi meaning Chinese words; undersea mine detection; texture analysis; three-dimensional object recognition; handwritten word recognition; and facial recognition.

2.14. Historical Background of Neural systems

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras.

Many important advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, relatively few researchers made important advances. These pioneers were able to develop convincing technology, which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers, and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding.

The history of neural networks that was described above can be divided into several periods:

1. First Attempts: There were some initial simulations using formal logic. McCulloch and Pitts (1943) developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons, which were considered to be binary devices with fixed thresholds. The results of their model were simple logic functions such as "a or b" and "a and b". Another attempt was by using computer simulations. Two groups (Farley and Clark, 1954; Rochester, Holland, Haibit and Duda, 1956). The first group (IBM researchers) maintained closed contact with neuroscientists at McGill University. So whenever their models did not work, they consulted the neuroscientists. This interaction established a multi disciplinary trend, which continues to the present day.

2. Promising & Emerging Technology: Not only was neuroscience influential in the development of neural networks, but psychologists and engineers also

contributed to the progress of neural network simulations. Rosenblatt (1958) stirred considerable interest and activity in the field when he designed and developed the Perceptron. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit.

3. Period of Frustration & Disrepute: In 1969 Minsky and Papert wrote a book in which they generalized the limitations of single layer Perceptrons to multi-layered systems. In the book they said: "...our intuitive judgment that the extension (to multi layer systems) is sterile". The significant result of their book was to eliminate funding for research with neural network simulations. The conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated.

4. Innovation: Although public interest and available funding were minimal, several researchers continued working to develop neuromorphically based computational methods for problems such as pattern recognition.

During this period several paradigms were generated which modern work continues to enhance. Grossberg's (Steve Grossberg and Gail Carpenter in 1988) influence founded a school of thought, which explores resonating algorithms. They developed the ART (Adaptive Resonance Theory) networks based on biologically plausible models. Anderson and Kohonen developed associative techniques independent of each other. Klopf (A. Henry Klopf) in 1972 developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called heterostasis.

Werbos (Paul Werbos 1974) developed and used the back-propagation learning method, however several years passed before this approach was popularized. Back-propagation nets are probably the most well known and widely applied of the neural networks today. In essence, the back-propagation net. Is a Perceptron with multiple layers, a different thresholds function in the artificial neuron, and a more robust and capable learning rule?

Amari (A. Shun-Ichi 1967) was involved with theoretical developments: he published a paper, which established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification. While Fukushima (F. Kunihiko) developed a stepwise trained multi layered neural network for interpretation of

handwritten characters. The original network was published in 1975 and was called the Cognitron.

5. Re-Emergence: Progress during the late 1970s and early 1980s was important to the re-emergence on interest in the neural network field. Several factors influenced this movement. For example, comprehensive books and conferences provided a forum for people in diverse fields with specialized technical languages, and the response to conferences and publications was quite positive. The news media picked up on the increased activity and tutorials helped disseminate the technology. Academic programs appeared and courses were introduced at most major Universities (in US and Europe).

Attention is now focused on funding levels throughout Europe, Japan and the US and as this funding becomes available, several new commercial with applications in industry and financial institutions are emerging.

6. Today: Significant progress has been made in the field of neural networks-enough to attract a great deal of attention and fund further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neutrally based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

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

The problem consists in constructing the controller for control of object (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.

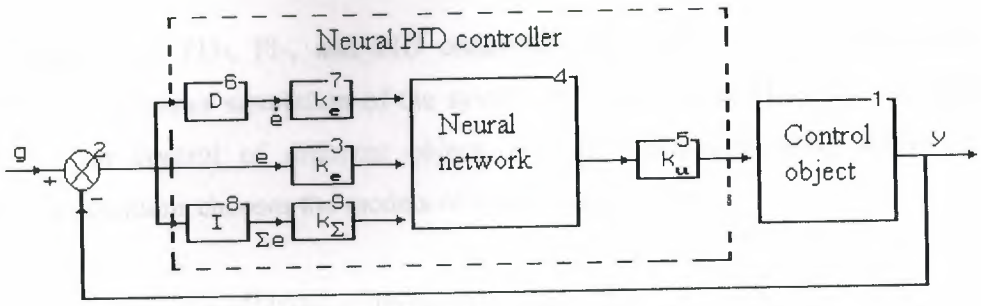

Figure 3.1. Structure of Neural PID- controller.

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

Assume there is target 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 achieving time response, which provides target step response of the system.

The input signals error e, error derivative e' and integral value of error $\int \mathbf{e(t)dt}$ 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 'back propagation' algorithm. The NN learning is performed in the closed control system, i.e. for learning NN error between target characteristic of control system and current output value of implemented system (output of control object) $\Delta(y, t)=k_e(g(t)-y(t))$ is used. That error is used for correction NN parameters for adjusting of controller.

Using learning algorithm of 'back propagation', the values of weight coefficients of NN is found.

3.2. Simulation of Neural Control Structure

Using neural PD-, PI-, and PID controllers for control of different object performs the computer simulation of the system by using neural PD-, PI-, and PID controllers for control of different object. For the simulation using following differential equations chooses the models of control object:

$$a_0 y^{(2)}(t)+a_1 y^{(1)}(t)+a_2 y(t)=b_0 u(t) \quad (3.2)$$

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.

$$a_0 y^{(2)}(t)+a_1 y^{(1)}(t)+a_2 y(t)=b_0 u(t-d) \quad (3.3)$$

Where $a_{0=}$ 6.3 min^2, a_1=11.2 min, a_2=1, b0=5.1 °C/(kgf/cm^2), d=2.5 min is delay;

$$a_0 y^{(3)}(t)+ a_1 y^{(2)}(t)+a_2 y^{(1)}(t)+a_3 y(t)=b_0 u(t-d) \quad (3.4)$$

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.3.2 (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 differentiation time Td=0.15 min., then transient process in the control system oscillates with 18% of transient overshoot. 8Where settling time t=1.3-1.5 min, static error $\varepsilon_{st}(\infty) \approx \mathbf{0.15x}(\infty)$, 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.

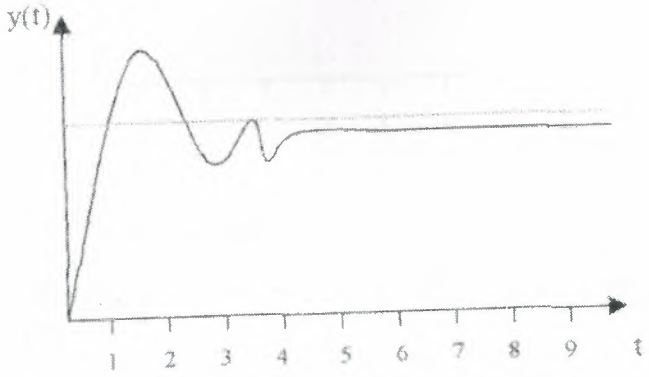

Figure 3.2. (a) PD Controller

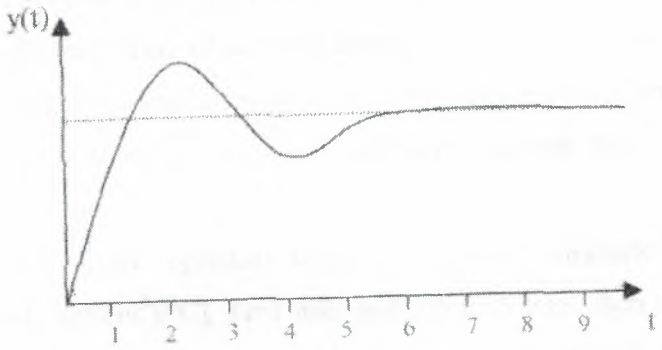

Figure 3.2. (b) PI Controller

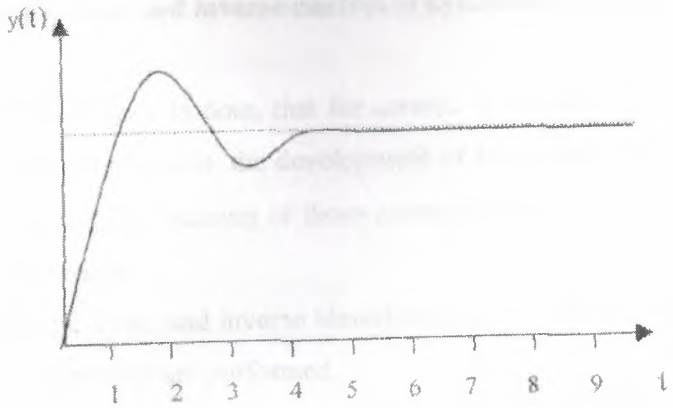

Figure 3.2. (c) PID Controller

Figure3.2.time response characteristics of control system with PD, PI and PID controller

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 figure.3.3. 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 'back propagation' algorithms is used. In the NN the following activation function is used.

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

In the inverse identification (figure.3.4) 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.

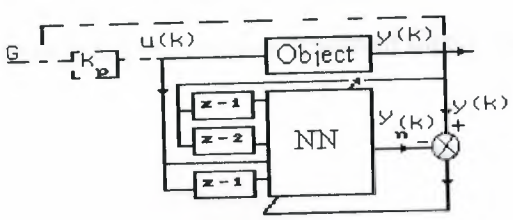

Figure 3.3 Structure of direct identification

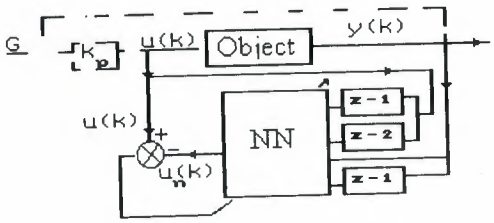

Figure 3.4. Structure of inverse Identification

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 object 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.

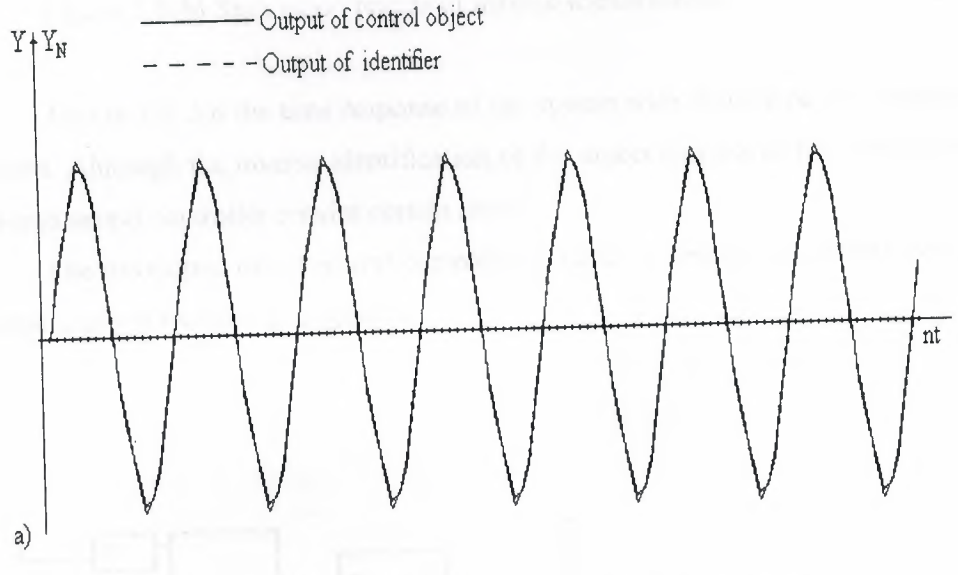

Figure 3.5(a) Simulation results of direct identification

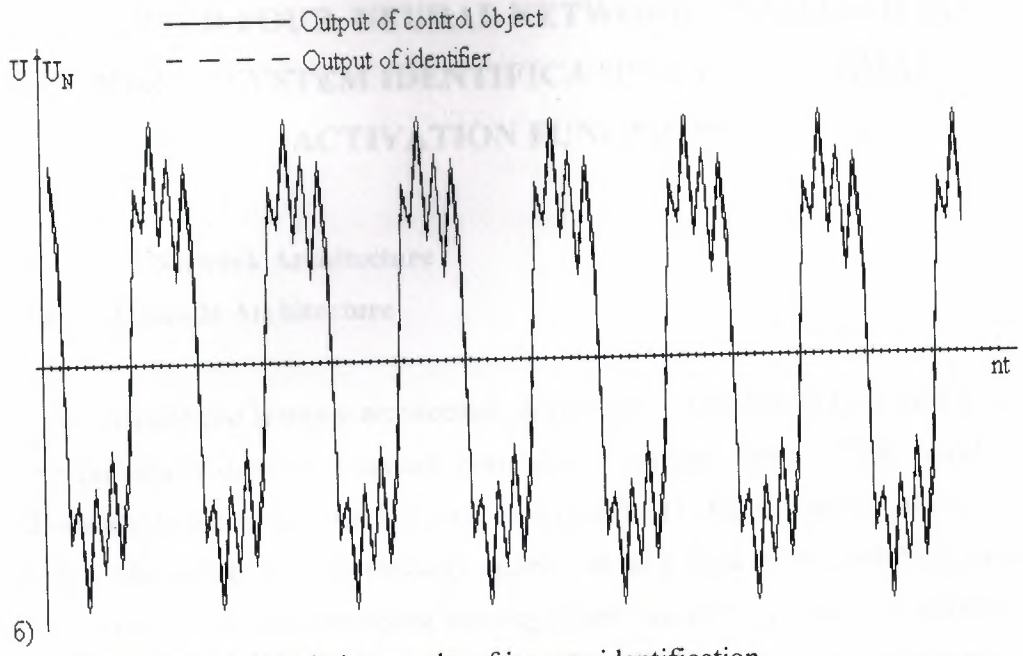

Figure 3.5(b) Simulation results of inverse identification

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.

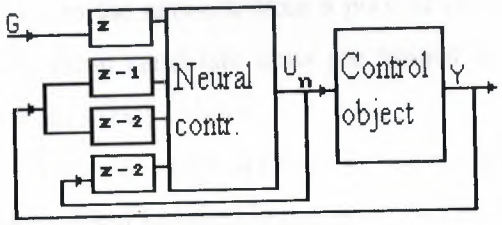

Figure 3.6. Structure of inverse controller.

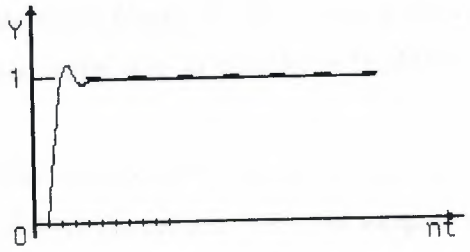

Figure 3.7 Time response of control system

CHAPTER FOUR. NEURAL NETWORK APPROACH TO CONTROL SYSTEM IDENTIFICATION WITH VARIABLE ACTIVATION FUNCTIONS

4.1. Neural Network Architecture

4.1.1. Cascade Architecture

The cascade two learning architecture, developed by Fahlman [14], is very similar to the previously developed cascade correlation algorithm. Both cascade correlation and cascade two combine the following two notions: (1) the cascade architecture, in which hidden units are automatically added one at a time to an initially minimal network, and (2) the accompanying learning algorithm, which creates and installs the new hidden units [13][14]. In cascade correlation, the learning algorithm attempts to maximize the magnitude of the correlation between the new hidden unit's output and the residual error signal. This covariance measure tends to overshoot small errors, however, and thus is not suitable for continuous-valued outputs. The cascade two algorithms correct this problem by attempting to minimize the sum-squared difference between the scaled unit outputs and the residual error [13]. Training proceeds as summarized below. Initially, there are no hidden units in the network, only the input/output connections. These weights are trained first, thereby capturing any linear relationship between the inputs and outputs. With no further appreciable decrease in the error measure (in cascade two, the sum squared error), the first hidden unit will be added to the network from a pool of candidate units. Using the quickprop algorithm [12], these candidate units are trained independently and in parallel with different random initial weights.

After no more appreciable error reduction occurs, the best candidate unit is selected and installed in the network. Once installed, the hidden unit input weights are frozen, while the weights to the output units are retrained. By freezing the input weights for all previous hidden units, each training cycle is equivalent to training a three-layer feed forward neural network with a single hidden unit.

This allows for much faster convergence of the weights during training than in a standard back prop network where many hidden unit weights are trained simultaneously.

The process is repeated until the algorithm succeeds in reducing the sum-squared error sufficiently for the training set or the number of hidden units reaches a specified maximum number. Note that each new hidden unit receives as input connections from all previous units, including all input units as well as previous hidden units. Figure 4.1 below illustrates how a 2-input, 1-output network grows as 2 hidden units are added. We believe that the cascade two architecture offers several advantages, particularly relevant for mapping of non-linear continuous-valued functions. First, the algorithm adjusts the architecture of the network automatically, thus obviating the need for a priori guessing of the necessary network architecture. Second, the cascade architecture can potentially model higher degrees of non-linearity with fewer hidden units than might be required in a single or two hidden-layer network. Finally, and perhaps most importantly, the incremental addition of hidden units allows for new hidden units to have variable activation functions.

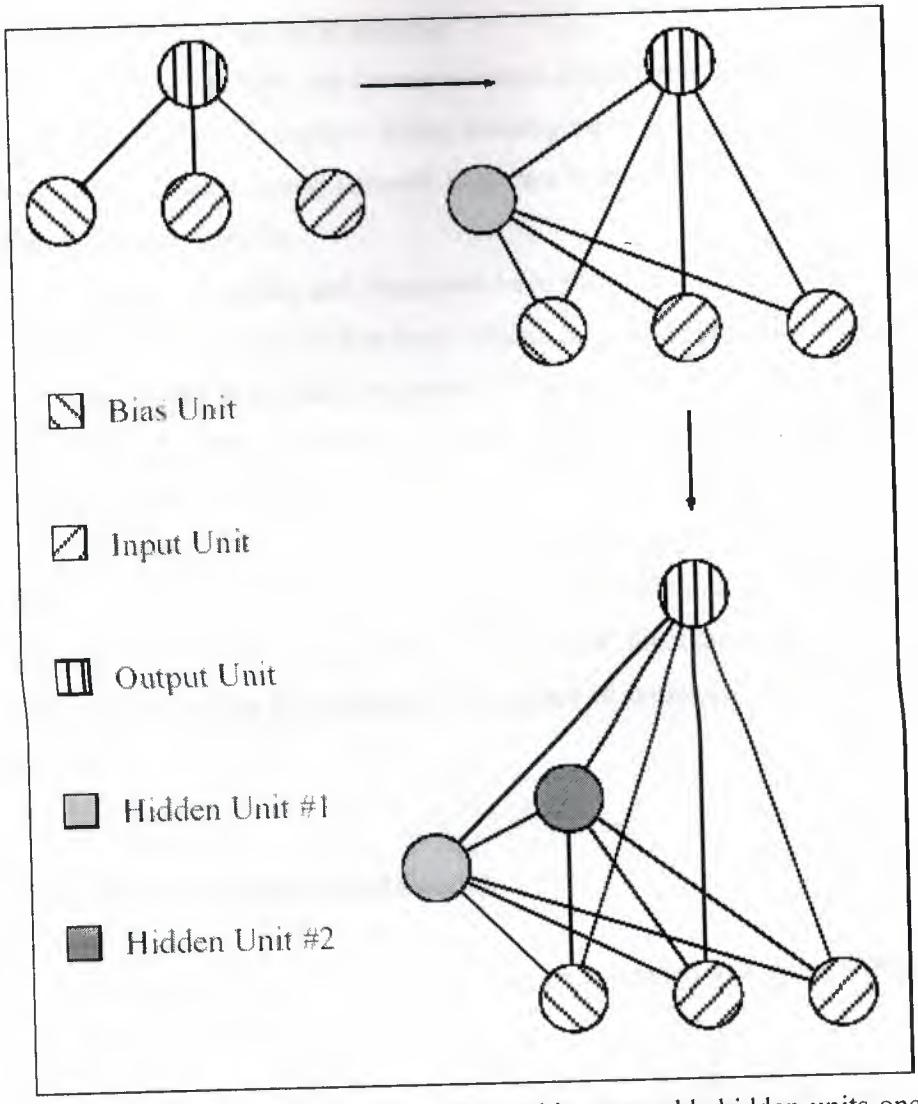

Figure 4.1. The cascade two learning architecture adds hidden units one at a time as shown in the above diagram. All connections are feed forward.

In the pool of candidate units, we can assign a different non-linear activation function to each unit. These functions can include but are not limited to the sigmoid function, sine or cosine functions, and the Gaussian function. Thus, if the function to be approximated has a strong sinusoidal dependence of some sort, it is more efficient to have one sinusoidal hidden unit rather than several sigmoidal units, which have to act together to first, approximate a sinusoidal dependence. During candidate training, the algorithm will select for instalment whichever candidate unit reduces the sum-squared error of the training data the most.

Hence, the unit with the most appropriate activation function at

That point during training is selected.

Finally, we note that the cascade two architecture is capable of arbitrary, non-linear function approximation. Using Kolmogorov's theorem, Kurkova shows in [19] that a feed forward neural network with two hidden layers is sufficient for arbitrary function approximation.

In fact, Cybenko and Funahashi have shown separately that a continuous feed forward neural network with a *single* hidden layer and sigmoidal activation functions can approximate non-linear mappings arbitrarily well [11][15]. Since any multilayer feed forward neural network with full connectivity between consecutive layers is simply a special case of a cascade network with an equal number of hidden units, these function approximation theorems extend trivially for this architecture. Furthermore, Cybenko shows that there is no strict theoretical argument for confining the activation functions exclusively to sigmoidal functions, and shows, for example, that *sine* and *cosine* are complete in the space of *n*-dimensional continuous functions [11].

4.1.2. Dynamic System Identification

In general, a dynamic system may be expressed as a finite difference equation of the general form,

$$\bar{y}(K+1) = g(\bar{y}(k), \bar{y}(k-1),..., \bar{y}(k-n), \bar{u}(k), \bar{u}(k-1),..., \bar{u}(k-m)) \quad \text{(Eq.1)}$$

Where $g(.)$ is some arbitrary non-linear function $\bar{y}(k)$, is the output vector and $\bar{u}(k)$ is the input vector at time step k. Most neural networks are only capable of static input/output mapping however, to overcome this problem, Narendra suggests providing a time history of data as input to the neural network [13][14]. Thus, *static* feed forward neural networks have the potential to approximate complex non-linear mappings of *dynamic* systems for which no analytic model may exists. For example, Figure 4.2 illustrates how this is done for a SISO system of the form,

$$y(k+1) = f(y(k), y(k-1), y(k-2), u(k), u(k-1)) \quad \textbf{(Eq.2)}$$

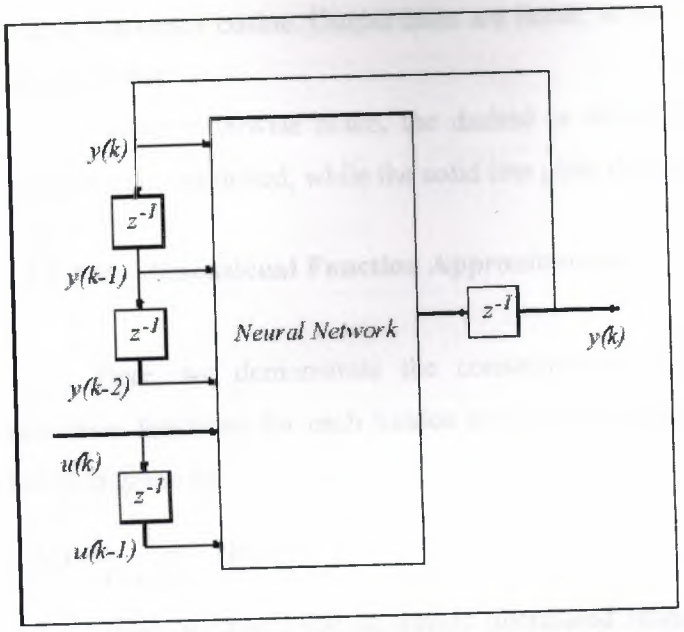

Figure 4. 2. The diagram illustrates how a dynamic system is mapped onto a static feed forward neural network.

In [24], special cases of (Eq.1) are classified depending on whether part of the relationship in the equation is linear. Since the cascade architecture begins with direct linear connections between inputs and outputs, such classification is unnecessary here.

4.2. Control System Modelling

Below, we present simulation results, which serve a three-fold purpose. First, the simulations demonstrate the feasibility and advantage of variable activation functions over *a* priori specification of activation functions. Second, the simulations demonstrate the neural network's ability to model dynamic systems from input/output data vectors. Finally, we show that the neural network can learn a known control strategy for a sample system. Such learning is crucial to identifying components of the human control process.

For all simulations, we allowed a maximum of 250 epochs to train the weights in a pool of eight candidate units. In the case of variable activation types, the pool of candidate units has the following function types: (1) standard symmetric sigmoid, with a (-0.5, 0.5) range, (2) standard zero-mean Gaussian, (3) Bessel function of the first kind of order zero and one, (4) sine, (5) cosine, (6) double frequency sine, and (7)

double frequency cosine. Output units are linear, so as to allow the outputs to assume any real value.

Unless otherwise noted, the dashed or dotted line in each figure shows the function to be modelled, while the solid line plots the output from the trained network.

4.2.1. One-dimensional Function Approximation

Here, we demonstrate the consequences of utilizing different non-linear activation functions for each hidden unit by modelling a simple, static one-variable function given by,

$$f(x) = \frac{1}{(1+x^2)} \quad \textbf{(Eq.3)}$$

Here, we use 1500 uniformly distributed random data points in the Interval $x \in [4,-4]$ as training data. We train two different networks, one with all sigmoidal units, and one with variable hidden units. In each case we stop training after six hidden units have been added. The hidden unit types in the network with variable activation functions follow in the order of insertion: (1) Bessel function of order zero, (2) sine, (3) double frequency cosine, (4) Bessel function of order one, (5) sine, and (6) double frequency cosine. Figure 4.3 below shows the network output for the variable-unit network, while Figure 4.4 shows the network output for the sigmoidal network. Figure 4.5 and Figure 4.6 show the approximation errors for Figure 4.3 and Figure 4.4, respectively. Table 1 summarizes the relationship between approximation error and activation functions for three different $f(x)$. In each case, the neural networks are trained to a size of six hidden units.

Table 1: Approximation Error for Various f(x)

$f(x)$	Sigmoidal Units (RMS error)	Variable Units (RMS error)
$0.6\sin(\pi x) + 0.3\sin(3\pi x) + 0.1\sin(5\pi x)$	0.0397	0.0097
$x^3 + 0.3x^2 - 0.4x$	0.0140	0.0069
$\frac{1}{(1+x^2)}$	0.0215	0.0045

4.2.2. Non-linear Difference Equation

Below, we compare our network architecture to a standard multiplayer feed forward back prop network as described by Narendra in [20]. The difference equation we want to approximate is given by,

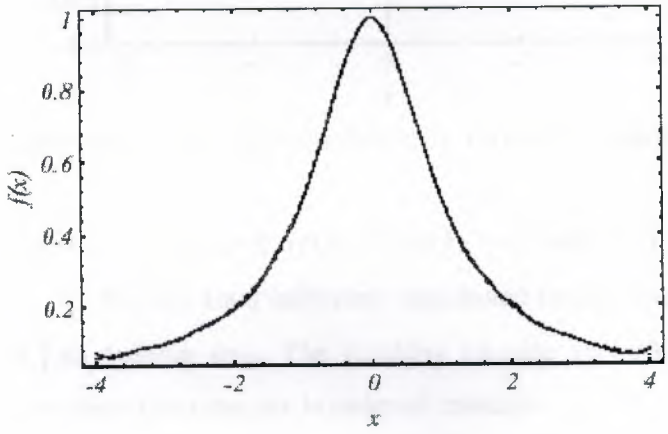

Figure 4.3. The network with non-sigmoid units performs better in approximating the function f (x).

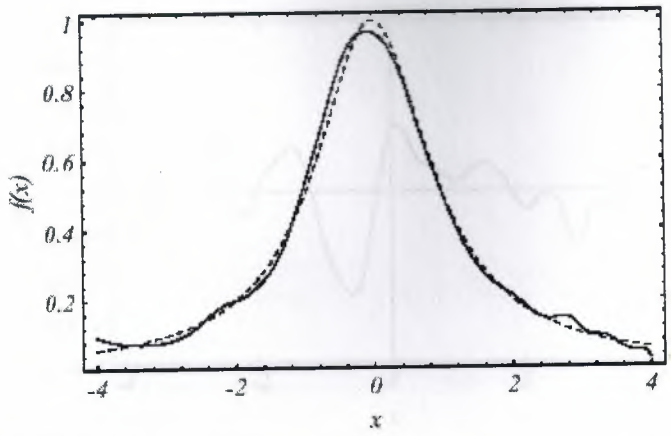

Figure. 4.4. The sigmoid network performs less well in approximating f (x).

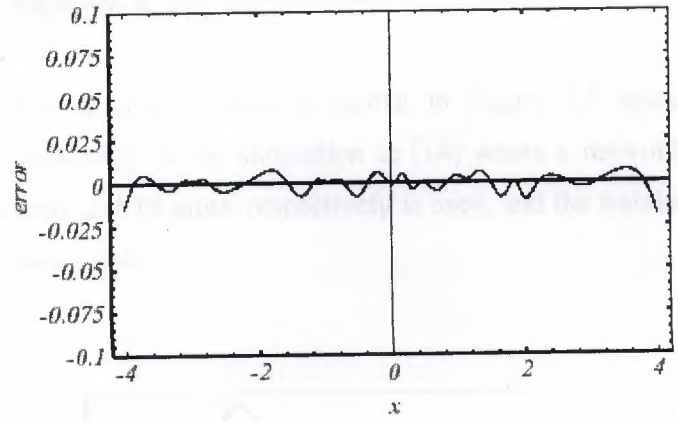

Figure.4. 5. The approximation error is relatively uniform over the training interval.

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + u(k)^3 + 0.3u(k)^2 \_ 0.4u(k) \quad \text{(Eq.4)}$$

We use 1000 uniformly distributed random inputs in the interval $u(k) \in [-1, 1]$ as training data. The resulting cascade network has three hidden units whose activation functions are in order of insertion:

(1) Double frequency cosine, (2) Bessel function of order one, and (3) sine.

To test the network, we use an input given by, $u(k) = \sin(\frac{2\pi k}{250})$ (Eq.5)

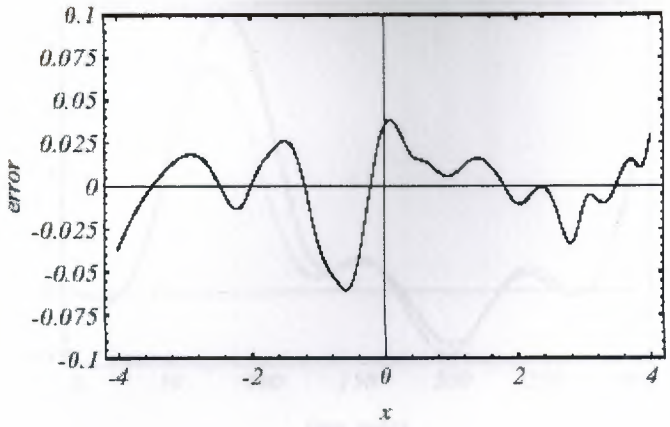

Figure.4. 6. The approximation error is much larger for the sigmoidal network.

The resulting output is shown in Figure 4.7 below. This result compares very favourably to the simulation in [14] where a network with two hidden layers of 20 units and 10 units, respectively is used, and the training data set includes over 50,000 data points.

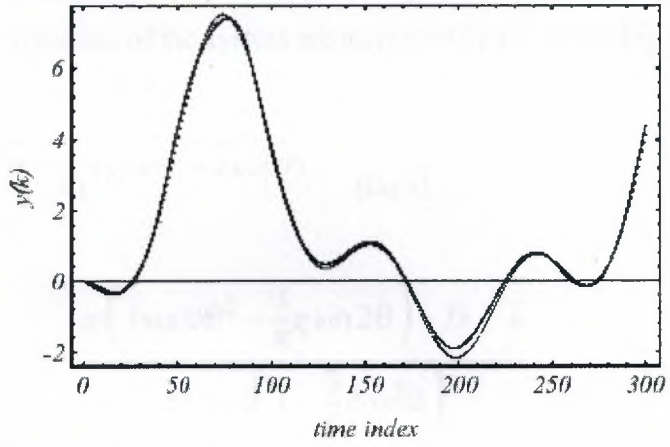

Figure.4.7. The 3-hidden unit neural network performs well in tracking the output of the dynamic system.

A network with three sigmoid hidden units performs significantly worse as is shown in Figure 4.8.

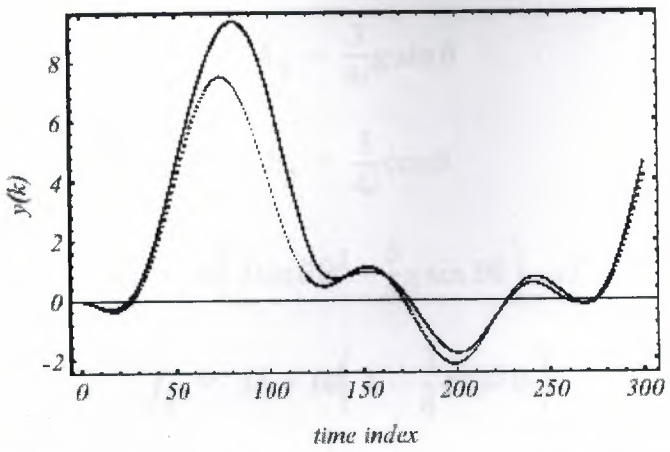

Figure. 4.8. More sigmoid hidden units are required to match the performance of the neural network in Figure4.7.

4.2.3. Control Application

Below, we simulate a non-linear controller for the classic inverted pendulum system. This is a traditional benchmark problem in control since the dynamics of the system are non-linear and coupled, and the open-loop system is unstable. The dynamics of the system are governed by the following equations [6]:

$$\ddot{\theta} = \frac{3}{41}(g\sin\theta - \ddot{x}\cos\theta) \qquad \text{(Eq.6)}$$

$$\ddot{x} = \frac{m\left(l\sin\theta\dot{\theta}^2 - \frac{3}{8}g\sin 2\theta\right) - f\dot{x} + u}{M + m\left(1 - \frac{3}{4}\cos^2\theta\right)} \qquad \text{(Eq. 7)}$$

We use the following non-linear control law as teacher to the neural network:

$$h_1 = \frac{3}{4l} g \sin\theta \qquad \text{(Eq. 8)}$$

$$h_2 = \frac{3}{4l} \cos\theta \qquad \text{(Eq. 9)}$$

$$f_1 = m\left(l \sin\theta \dot{\theta}^2 - \frac{3}{8} g \sin 2\theta \right) - f\dot{x} \qquad \text{(Eq. 10)}$$

$$f_2 = M + m\left(1 - \frac{3}{4} \cos^2\theta \right) \qquad \text{(Eq. 11)}$$

$$u = \frac{f_2}{h_2}\left[h_1 + k_1 (\theta - \theta_d) + k_2 \dot{\theta} + c_1 (x - x_d) + c_2 \dot{x} \right] - f_1 \qquad \text{(Eq. 12)}$$

For the simulations, we used the following numeric values:
M=1 kg, m=0.1 kg, l=1 m, f=5 kg/s, g=9.81 m/s2, k1=25, k2=10 ,$C1$=1,$c2$=2.6 Also we set x_d=0 m, $\theta_d = 0$ rad , which are the desired position of the cart and angle of the pendulum respectively. For details on all the parameters see [16]. This system is simulated numerically using Euler's approximation method with a time step of T = 0.02 seconds. The neural network takes as input the current and previous x positions, as well as the current and previous positions. It is trained to approximate the control law given in (Eq.12). As training data, we generate 500 uniformly distributed random input/output vectors in the following range:

$$x \in [-1, 1] \quad \text{(m)} \qquad \text{(Eq. 13)}$$

$$\theta \in [-0.5, 0.5] \quad \text{(rad)} \qquad \text{(Eq. 14)}$$

$$(x_{current} - x_{previous}) \in [-0.04, 0.04] \quad \text{(m)} \qquad \text{(Eq. 15)}$$

$$(\theta_{current} - \theta_{previous}) \in [-0.02, 0.02] \quad \text{(rad)} \qquad \text{(Eq. 16)}$$

We found that as few as three hidden units were sufficient to model the controller, even for large initial values of θ . Below we compare the performance of a

trained neural network with three sinusoidal hidden units to that of the actual control law. The initial

Conditions for the simulation are,

$$[x, \dot{x}, \theta, \dot{\theta}] = [0, 0, 0.6\,(rad), 0] \qquad \text{(Eq. 17)}$$

Note that the initial condition for is outside the range of θ the training data. Figure4.9 compares the actual controller and the neural network controller performance. The controller and neural network generate virtually identical results. Figure4.10 shows the difference in response between the actual and the neural network controller.

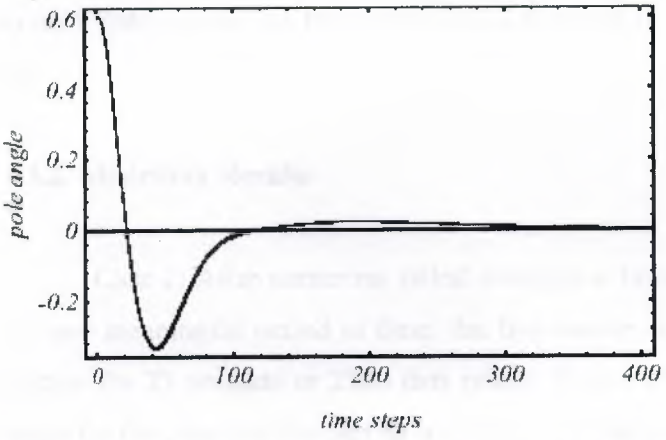

Figure 4.9. The angle of the pendulum is controlled almost identically for the non-linear control law and the neural network controller.

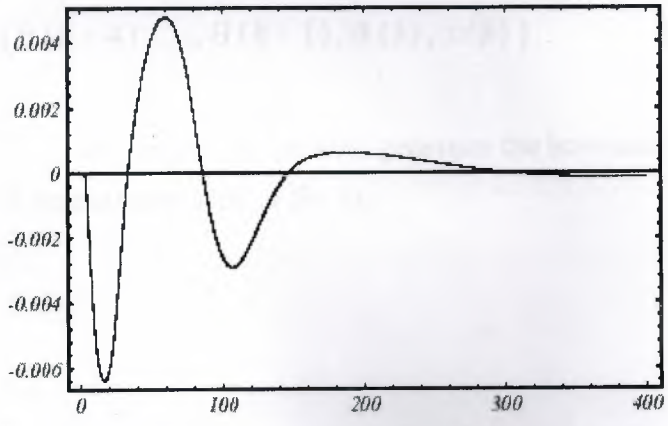

Figure4.10. This figure plots the error between the angle position caused by the non-linear control law and the angle position caused by the neural network controller.

4.3. Modelling Human Control Strategy

4.3.1. Experimental Set-up

In this section, we show preliminary results in modelling human control strategy. For the experiment, a human subject is shown an inverted pendulum-cart system on a computer screen, and is able to control the horizontal force to be applied to the cart via the horizontal mouse position. The parameters for this cart-pendulum system are equivalent to those given in the previous section. Thus, we have replaced the non-linear control law with a human being as teacher for the neural network. The system state, as well as the control input provided by the human, is recorded at 100 Hz.

4.3.2. Modelling Results

Case 1: After numerous failed attempts at keeping the pendulum from falling for any meaningful period of time, the first human subject successfully controls the system for 23 seconds or 2300 data points. Figure 4.11 below shows the pendulum angle for the time that the human is able to keep the pendulum from falling. From this data, 750 randomly selected data points are selected to train the network, while another 750 randomly selected data points are used for cross validation. The neural network to be trained from this data takes six inputs, namely, the past five values of the pendulum angle, as well as the velocity of the cart,

$$\{\theta(k-4), \ldots, \theta(k-1), \theta(k), \dot{x}(k)\} . \qquad \text{(Eq. 18)}$$

As output, the network generates the horizontal force to be applied to the cart in the next time step, $.u\ (k+1)$.

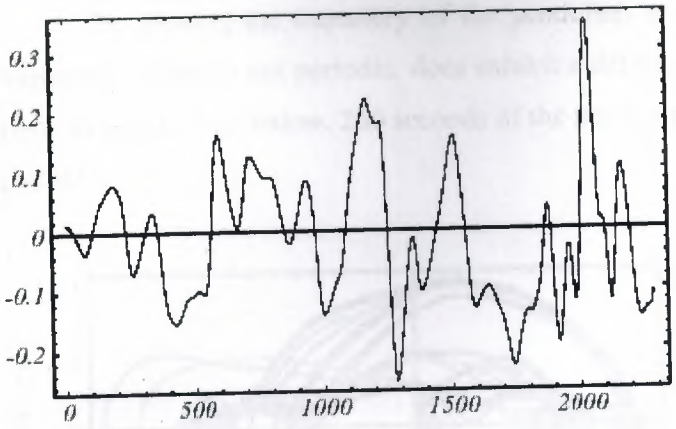

Figure 4.11. Data from this run of 23 seconds was used to train the neural network to control the inverted cart-pendulum system.

Here, the pendulum angle is shown in radians.

We allow a maximum of 150 epochs to train the weights as each new hidden unit is added. Here, all hidden unit activation function types are one of the sinusoidal functions; we stop training with twelve hidden units. Figure 4.12 below shows the resulting neural network control of the pendulum-cart system with $\theta initial$ =0.2 for 20 seconds.

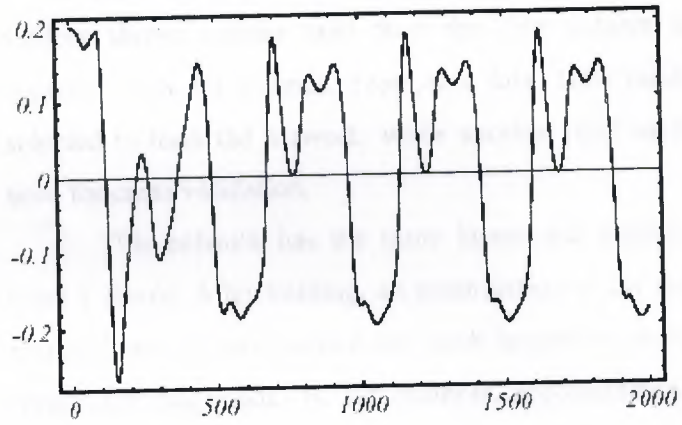

Figure 4.12. Neural network control of the inverted pendulum-cart system. Here, the pendulum angle is shown in radians.

By plotting the trajectory of the pendulum in phase space, we see that the trajectory, although not periodic, does exhibit a definite pattern over a long period of time. In Figure 4.13 below, 200 seconds of the pendulum trajectory in phase space are plotted.

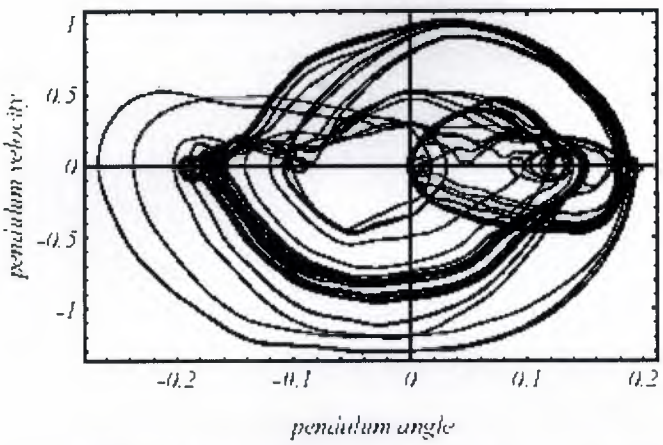

Figure 4.13. Pendulum trajectory in phase space.

It was determined experimentally that this neural network controller is stable for -0.92 <θ*initial*< 0.98.

Case 2: A different human subject is also asked to control the system. This subject shows greater skill than the first subject and has a successful run of approximately 60 seconds. From this data, 1000 randomly selected data points are selected to train the network, while another 1000 randomly selected data points are used for cross validation,

This network has the same inputs and outputs, and training proceeds as in Case 1 above. After training, an examination of the resulting weights in the network revealed several weights that are much larger than many smaller weights. The largest weight (in magnitude) is, for example, approximately 1201. Thus, all weights less than six in magnitude are set to zero. The resulting controller proves to be remarkably simple, and can be expressed by,

$$u(k+1) = w_1\theta(k) + w_2\theta(k-2) + w_3\theta(k-4) + w_4\dot{x} \qquad \text{(Eq. 19)}$$

Where $w1$=-350, w2=1201, w3= -925 and w4=7.6. Thus a traditional linear feed back controller has been abstracted from training data provided by human operator. This controller is stable for $-1.04 < \theta initial < 1.05$.

CONCLUSION

Learning control from humans by example is an important concept for making controllers, machines more intelligent. Neural networks are well suited to generate the complex nonlinear mapping of the human control process, which maps sensory inputs to control action outputs.

The work have been described in the neural network of intelligent structure is focusing on the learning aspect of smart structure controllers with neural architectures along two main research directions of The basic research effort that aims at the development of novel neural control architectures.

The neural network controller is usually structured according to the neural network plant model using external feed back of the control signal and delayed values of the commanded input using time delay lines.

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

The encouragement results for nonlinear continuous function mapping and dynamic system identification by utilizing new neural network architecture are presented.

The recurrent network architecture and learning Process is well suited for efficiently mapping continuous nonlinear functions.

The method allows a neural network to learn both a known nonlinear, coupled control law, as well as unknown nonlinear human control strategy.

REFERENCES

[1] Brückner, S. and Rudolph, S.: "Neural Networks Applied to Smart Structure Control." Proceedings SPIE Aerosense 2000 Conference on Applications and Science of Computational Intelligence III, Orlando, FL, 24. -28. 4. 2000.

[2] Brückner, S. and Rudolph, S.: "Neural Control and System Identification Using a Similarity Approach." Proceedings SPIE 7th International Symposium on Smart Structures and Materials, Newport Beach, California, March 5-9th, 2000.

[3] Gunaratnam, D. and Gero, J.: "Effect of Representation on the Performance of Neural Networks in Structural Engineering Applications." Microcomputers in Civil Engineering 9, 97-108, 1994.

[4] Hunt, K., Sbarbaro, D., Zbikowski, R.and Gawthrop, P.: "Neural Networks for Control Systems - a Survey," Automatica 28 (6), 1083-1112, 1992.

[5] Narendra, K.: "Adaptive control using neural networks," in: "Neural Networks for Control", Miller, T., Sutton, R. and Werbos, P. (eds.), MIT Press, Cambridge, MA, chapter 5, 115-142, 1990.

[6] Rudolph, S.: "On Topology, Size and Generalization in Non-Linear Feed-Forward Neural Networks." Neurocomputing 16 (1), 1-22, 1997.

[7] Schneider, G., Korte, D. and Rudolph, S.: "Neural Network Correspondences of Engineering Principles." Proceedings SPIE Aerosense 2000 Conference on Applications and Science of Computational Intelligence III, Orlando, FL, 24. -28. 4. 2000.

[8] McGraw-Hill P.liang 1996 "Neural network fundamentals with graphs,algorithim,and applications"

[19] Kurkova, V., "Kolmogorov's Theorem and Multilayer Neural Networks," *Neural Net.*, Vol. 5, No. 3, pp. 501-506, 1992.

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

[21] Narendra, K. S., Parthasarathy, K., "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, pp. 4-27, 1990.

[22] The 1989 Neuro Computing Bibliography.

[23] Neural Network Study (October, 1987-February, 1989).

[24] MIT Lincoln Lab. Neural Networks, Eric Davalo and Patrick Naim. Prof. Aleksander. articles and Books. (from Imperial College)

[25] R.H.Abiev.Controllers Based on Neural Networks//Uchenie zapiski,AzGNA,1994,8897pp.(Russian)

[26] R.H.Abiev,K.W.Bonfig,F.T.Aliev.controller based on fuzzy neural network for technological process.//ICAFS-96,Siegen,Germany

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