NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

RECURRENT NEURAL NETWORK FOR CONTROL OF DYNAMIC PLANTS

Graduation Project
COM- 400

Student: Atayeb Al-atawneh (20000485)

Supervisor: Assoc.Prof.Dr.Rahib Abiyev

Nicosia - 2005

ACKNOWLEDGMENT

I want to thank Assoc.Prof.Dr. Rahib Abiyev to be my advisor, under his guidance; I successfully overcome many difficulties and learn a lot about Recurrent Neural Network. In each discussion he explained my questions patiently, and I felt my quick progress from his advices. He always helps me a lot either in my study or in my life. I asked him many question in Computer and Neural Networks and he always answered my question quickly and in details.

I also want to thank my friends for there help: Omar Alatawneh, Ibrahem Alatawneh, Saleem Alatawneh. Also my NEU friends: my best friend Zaki Abdallah, Nael Attoun, Tariq Alkhalidi, Fadi Asadi, Ismael ALhams and Ahmad Abdallah. Being with them make my time full of fun.

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

ABSTRACT

So far we have been studying neural networks that only have forward connections and thus there have been no loops. Networks with loops - recurrent connections is the technical term used in the literature have an internal dynamics.

The neural networks that we have learned so far have only retained information about the past in terms of their weights, which have been changed according to the learning rules.

Recurrent networks also retain information about the past in terms of activation levels, activation is preserved for a certain amount of time: it is "passed back" through the recurrent connections. This leads to highly interesting behaviors, behaviors that can be characterized in terms of dynamical systems.

In this project recurrent neural network and it's learning algorithim is applied for identification and control of dynamic plants, the obtained results demonstrate the efficiency of usage of recurrent neural network in control of dynamic plants.

TABEL OF CONTENTS

INTRODUCTION

Recurrent neural networks have been an important focus of research and development during the 1990's. They are designed to learn sequential or timevarying patterns. A recurrent net is a neural network with feedback (closed loop) connections [Fausett, 1994]. Examples include BAM, Hopfield, Boltzmann machine, and recurrent backpropagation nets [Hecht-Nielsen, 1990]. Recurrent neural network techniques have been applied to a wide variety of problems. Simple partially recurrent neural networks were introduced in the late 1980's by several researchers including Rumelhart, Hinton, and Williams [Rummelhart, 1986] to learn strings of characters. Many other applications have addressed problems involving dynamical systems with time sequences of events.

In recent years neural network research has received a steadily growing interest from a large number of different scientific disciplines. Among these are for instance computational neuroscience, statistical physics, distributed processing, pattern recognition, image analysis, or control theory. Work on neural networks is often jointly carried out by scientists with various backgrounds and establishes connections between different fields. Consequently, it has been a characteristic of neural network analysis that a large variety of mathematical tools and methods are used, including such different approaches as statistics, group theory, statistical mechanics, probability theory, linear algebra, Lyapunov functions, dynamical systems theory, combinatorics, or approximation theory.

This multidisciplinary atmosphere has proven to be very inspiring and led to a fairly deep grounding of neural network theory in classical theoretical approaches as well as to numerous application examples, and, increasingly important, to industrial applications. With the present work we contribute to the theoretical understanding of the behaviour of dynamical neural networks by introducing a system-theoretic oriented point of view. Much in the spirit described above we use a number of well founded concepts from non-linear system theory along with recently developed numerical methods to approach the problem of input-output stability of recurrent neural networks.Though many of the methods used are classical, their connection and joint application in the neural network domain yields a number of new results.

In the converse direction it turns out that we have to extend the known methodology at certain points to get reasonable results for recurrent networks. However, these new methods are not restricted to network theory and hold for more general systems as well, in particular we present a new method to compute input-output gains for general feedback systems, and we present an entirely new approach to stability of systems with interval bounded parameters. Thus the application of system theory in the neural network domain leads to contributions to both fields, system theory and neural network research.

My first Chapter is all about the introduction of neural network and my second chapter is about structure of recurrent neural network and my third chapter about training recurrent network for control and my last chapter is application of recurrent neural network for control .

CHAPTER ONE

1. INTRODUCTION TO NEURAL NETWORK

1.1 Overview

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 even with the intelligence of an earthworm. Imagine the power of the machine, which has the abilities of both computers and human.

It would be the most remarkable thing ever. All humans can live happily ever after. This is the aim of artificial intelligence in general. Neural network simulation appears 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 emulation. The future of neural network is wide open, and may lead to many answers and many questions.

Neural network process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural network learn by example. They cannot be programmed to perform a specific task. The examples must be clear.

1.2 Neural Network

In information technology, a neural network is a system of programs and data structure that approximate the operation of the human brain. A neural network usually involves a large number of processors operating in parallel, each with its own small sphere of knowledge and access to data in its local memory. Typically, a neural network is initially "trained" or fed large amounts of data relationships (for example, "a grandfather is older than a person's father"). A program can tell the network how to behave in response to an external stimulus (for example, to input from a computer user who is interacting with the network) or can initiate activity on its own (within the limits of its access to the external world).

In making determinations, neural networks use several principles, including gradient-based training, fuzzy logic, genetic algorithms, and Bayesian methods. Neural networks are sometimes described in items of knowledge layers, with, in general, more complex networks having deeper layers. In feed-forward systems, learned relationships about data can "feed forward" to higher layers of knowledge. Neural networks can also learn temporal concepts and have been widely used in signal processing and time series analysis.

Current applications of neural networks include: oil exploration data analysis, weather prediction, the interpretation of nucleotide sequences in biology labs, and the exploration of models of thinking and consciousness. In his novel, Galatea [1], Richard Powers envisioned a neural network (named "Helen") that could be thought to pass a comprehensive exam in English literature.

1.3 The Sudden Rise of Neurocomputing

The majority of information processing today is carried out by digital computers. This has led to the widely held misperception that information processing is dependent on digital computers. However, if we look at cybernetics and the other disciplines that form the basis of information science, we see that information processing originates with living creatures in their struggle to survive in their environments, and that the information being processed by computers today accounts for only a small part - the automated portion - of this. Viewed in this light, we can begin to consider the possibility of information processing devices that differ from conventional computers. In fact, research aimed at realizing a variety of different types of information processing devices is already being carried out, albeit in the shadows of the major successes achieved in the realm of digital computers. One direction that this research is taking is toward the development of an information processing device that mimics the structures and operating principles found in the information processing systems possessed by humans and other living creatures.

Digital computers developed rapidly in and after the late 1940's, and after originally being applied to the field of mathematical computations, have found expanded applications in a variety of areas, to include text (word), symbol, image and voice processing, i.e. pattern information processing, robot control and artificial intelligence. However, the fundamental structure of digital computers is based on the principle of sequential (serial) processing, which has little if anything in common with the human nervous system.

The human nervous system, it is now known, consists of an extremely large number of nerve cells, or neurons, which operate in parallel to process various types of information. By taking a hint from the structure of the human nervous system, we should be able to build a new type of advanced parallel information processing device.

In addition to the increasingly large volumes of data that we must process as a result of recent developments in sensor technology and the progress of information technology, there is also a growing requirement to simultaneously gather and process huge amounts of data from multiple sensors and other sources. This situation is creating a need in various fields to switch from conventional computers that process information sequentially, to parallel computers equipped with multiple processing elements aligned to operate in parallel to process information.

Besides the social requirements just cited, a number of other factors have been at work during the 1980's to prompt research on new forms of information processing devices. For instance, recent neurophysiological experiments have shed considerable light on the structure of the brain, and even in fields such as cognitive science, which study human information processing processes at the macro level, we are beginning to see proposals for models that call for multiple processing elements aligned to operate in parallel. Research in the fields of mathematical science and physics is also concentrating more on the mathematical analysis of systems comprising multiple elements that interact in complex ways. These factors gave birth to a major research trend aimed at clarifying the structures and operating principles inherent in the information processing systems of human beings and other animals, and constructing an information processing device based on these structures and operating principles.

1.4 The Biological Foundation of NeuroComputing

Neurocomputing involves processing information by means of changing the states of networks formed by interconnecting extremely large numbers of simple processing elements, which interact with one another by exchanging signals. Networks such as the one just described are called artificial neural networks (ANNs), in the sense that they represent simplified models of natural nerve or neural networks(figure 1.1.).

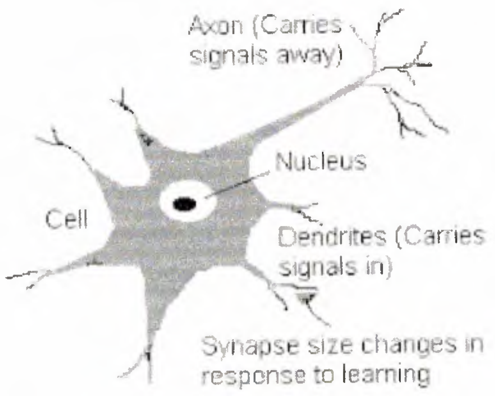

Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carries signals in)
Synapse size changes in response to learning

Figure 1.1. A simple neuron cell

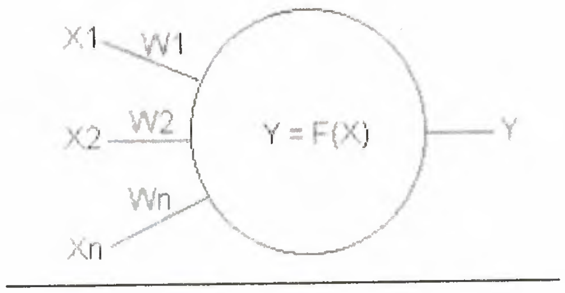


Figure 1.2. A schematic diagram of a neuron

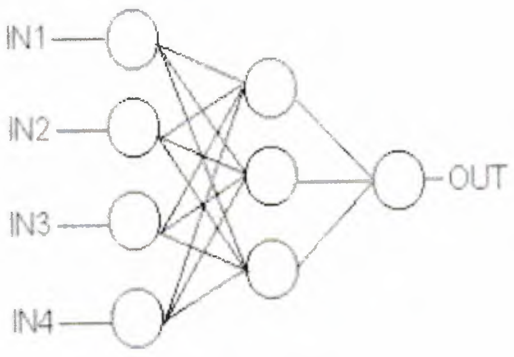


Figure1.3. A feed forward neural network

The basic processing element in the nervous system is the neuron. The human brain is composed of about 1011 of over 100 types. Tree-like networks of nerve fiber called dendrites are connected to the cell body or soma, where the cell nucleus is located. Extending from the cell body is a single long fiber called the axon, which eventually branches into strands and substrands, and are connected to other neurons through synaptic junctions, or synapses.

The transmission of signals from one neuron to another at a synapses is a complex chemical process in which specific transmitter substances are released from the sending end of the junction. The effect is to raise to lower the electrical potential inside the body of the receiving cell. If the potential reaches a threshold, a pulse is sent down the axon - we then say the cell has "fired".

In a simplified mathematical model of the neuron, the effects of the synapses are represented by "weights" which modulates the effect of the associated input signals, and the nonlinear characteristics exhibited by neurons is represented by a transfer function which is usually the sigmoid function. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm, usually by a small amount *Wj = **Xj where * is called the learning rate and * the momentum rate.

1.5 Building A Neural Network

Since 1958, when psychologist Frank Rosenblatt [2] proposed the "Perceptron," a pattern recognition device with learning capabilities, the hierarchical neural network has been the most widely studied form of network structure. A hierarchical neural network is one that links multiple neurons together hierarchically, as shown in Figure 1.3. The special characteristic of this type of network is its simple dynamics. That is, when a signal is input into the input layer, it is propagated to the next layer by the interconnections between the neurons. Simple processing is performed on this signal by the neurons of the receiving layer prior to its being propagated on to the next layer. This process is repeated until the signal reaches the output layer completing the processing process for that signal.

The manner in which the various neurons in the intermediary (hidden) layers process the input signal will determine the kind of output signal it becomes (how it is transformed).

As wee can see, then, hierarchical network dynamics are determined by the weight and threshold parameters of each of their units. If input signals can be transformed to the proper output signals by adjusting these values (parameters), then hierarchical networks can be used effectively to perform information processing.

Since it is difficult to accurately determine multiple parameter values, a learning method is employed. This involves creating a network that randomly determines parameter values. This network is then used to carry out input-to-output transformations for actual problems. The correct final parameters are obtained by properly modifying the parameters in accordance with the errors that the network makes in the process. Quite a few such learning methods have been proposed. Probably the most representative of these is the error back-propagation learning method proposed by D. E. Rumelhart et al. in 1986 [3]. This learning method has played a major role in the recent neurocomputing boom.

The back-propagation paradigm has been tested in numerous applications including bond rating, mortgage application evaluation, protein structure determination, backgammon playing, and handwritten digit recognition. Choosing the right methodology, or backpropagation algorithm, is another important consideration. In working with the financial applications, many have found that the back-propagation algorithm can be very slow. Without using advanced learning techniques to speed the process up, it is hard to effectively apply backpropagation to real-world problems. Overfitting of a neural network model is another area which can cause beginners difficulty. Overfitting happens when an ANN model is trained on one set of data, and it learns that data too well. This may cause the model to have poor generalization abilities - the model may instead give quite poor results for other sets of data.

1.6 The Analogy to the Brain

The most basic components of neural networks are modeled after the structure of the brain. Some neural network structures are not closely to the brain and some does not have a biological counterpart in the brain. However, neural networks have a strong similarity to the biological brain and therefore a great deal of the terminology is borrowed from neuroscience.

1.7 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(see figure 1.4.), each of these neurons can connect with up to 200000 other neurons. The power of the brain comes from the numbers of these basic components and the multiple connections between them.

All natural neurons have four basic components, which are dendrites, soma, axon, and synapses. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then output the final result. The figure below shows a simplified biological neuron and the relationship of its four components.

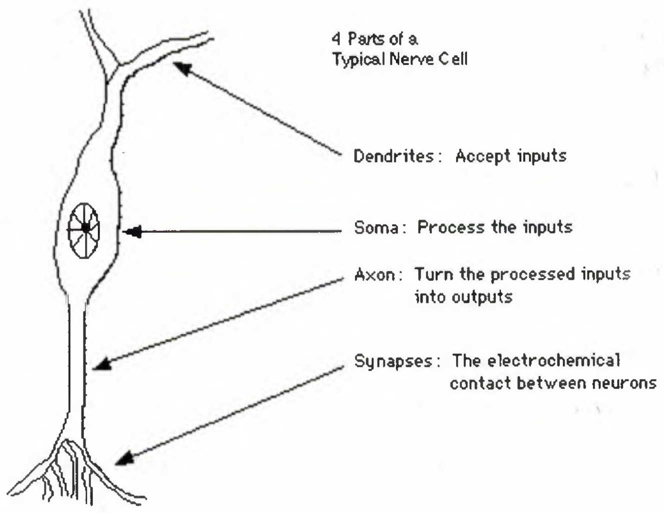

4 Parts of a Typical Nerve Cell
Dendrites: Accept inputs
Soma: Process the inputs
Axon: Turn the processed inputs into outputs
Synapses: The electrochemical contact between neurons

Figure1.4. The biological neuron

1.8 The Animal Brain

Much is now known about the animal brain, but so much more have to be learned. The partitioning of the brain into a left and right hemisphere, and into various subcomponents such as the cerebrum, pituitary gland, thalamus, hypothalamus, hippocampus, cerebellum, hindbrain, mid brain, are all known. Also, the general functions of the various regions of the human brain have been mapped out, e.g., the back regions for vision, the frontal lobe for logic and higher level thought processes, certain specific region for auditory perception, another region for soma to-sensory perception, etc.

What are less known are the specific electro-chemical interactions among neurons and the various neurophysiological processes involved in short-term and long term memory, not to mention even vaguer concepts such as “consciousness” and “intuition”.

There is a tight relationship between the weight and size of the brain and the intellectual capacity of the animal. The human brain is the largest among all animal brains. A chimpanzee’s brain is big compared to that of other animals, regardless of their relative weight, but lacks some parts of the human brain, such as the frontal lobe, which is associated with man’s logic and reasoning. Lower forms of animals in the animal kingdom structure that exhibit limited intelligence have smaller brains.

The cerebral cortex is the largest part of the brain and is composed of numerous interconnected neurons arranged in very thin sheets that are highly convoluted. There are approximately 10^{11} neurons in the human brain, and each is connected to a thousand to up to 10,000 other neurons. The amount of interconnectivity in the average human brain is therefore in the order of 10^{14} or 10^{15}!

The brain is mainly composed of massively interconnected cells called neurons. A typical neuron has a cell body called the soma, and root-like structures through which input signals are received, called dendrites. These dendrites connect to other neurons through what is called a synapse. Signals collected through the numerous dendrites are accumulated in the soma. If the accumulated signal exceeds the neuron’s threshold, then the cell is activated. This results in an electrical spike that is sent down the output channel called the axon. Otherwise, the cell remains inactive.

The efficiency by which signals from one neuron is sent to another neuron via a particular synapse is called the synaptic efficiency. Synapses are truly channels through which electrical impulses cross. These electrical impulses are regulated by chemicals known as neuro-transmitters.

The nature of the synaptic membranes and the chemical composition of neuro-transmitters determine whether a synapse is inhibitory or excitatory. When the synapse is excitatory, then the signal from the sending neuron tends to activate the neuron to which it is sent. Inhibitory synapses tend to prevent the activation of the neuron to which a signal is sent.

The strength of the electric signal that finally reaches the soma of the receiving neuron therefore depends on the strength of the original signal as well as on the efficiency of the synapse through which it will pass. The nature of the signal (inhibitory or excitatory) depends on the kind of neuro-transmitters found in the specific synaptic channel and on the type of synaptic membrane. In artificial neurons, we use real-valued connection weights to model the synaptic efficiency between two neurons, and their sign (positive/negative) indicate whether they are excitatory or inhibitory.

1.9 Why use a neural network?

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

Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.

Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manifactured which take advantage of this capability.

Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilites may be retained even with major network damage.

1.10 Some Real Life Applications

ANNs can be regarded, in one respect, as multivariate nonlinear analytical tools, and are known to be very good at recognizing patterns from noisy, complex data, and estimating their nonlinear relationships. Many studies have shown that ANNs have the capability to learn the underlying mechanics of the time series, or, in the case of trading applications, the market dynamics. In general, ANNs are known to possess these capabilities:

A number of development projects involving ANN technology have been publicized in the media recently. For example, Nippon Steel Corp [4]. has built a blast furnace operation control support system that makes use of ANNs. The neural network employed in this system has been equipped with functions that enable it to learn the relationship between sensor data and the eight kinds of temperature distribution patterns known from experience to pertain to the overall operation of blast furnaces, and to instantaneously recognize and output that pattern which most closely approximates sensor data input into the system. The neural network learns very quickly, and achieves a better than 90% pattern recognition ratio following learning. Since this system has been performing extremely well during operational testing, Nippon Steel is planning to introduce it into other aspects of its operations in addition to blast furnace control, to include the diagnosis of malfunctions, and other control processes.

A second example is the experimental work started by Daiwa Securities Co. , Ltd. and NEC [5] Corporation on applying neural network technology to the learning and recognition of stock price chart patterns for use in stock price forecasting. NEC had already developed neural network simulation software for use on its EWS 4800 series of workstations, and, by limiting stock price chart pattern learning to a few dozen major stocks, has improved the accuracy of this software's forecasting capabilities. Based on these results, the Daiwa Computer Services Co., Ltd., an information processing subsidiary of the Daiwa Securities Group, transferred the NEC system to its supercomputer and taught it to recognize the stock price chart patterns for 1,134 companies listed on the Tokyo Stock Exchange. DCS has since been putting this system to good use in the performance of stock price forecasting.

Mitsubishi Electric has combined neural network technology with optical technology to achieve the world's first basic optical neurocomputer system capable of recognizing the 26 letters of the alphabet. The system comprises a set of light-emitting diodes (LED) that output letter patterns as optical signals, optical fibers, liquid crystal displays (LCD) that display letter patterns and light receiving devices that read these letters. When letter data is input into this system, light emitted from the LEDs is input to the light receiving devices through the LCDs. At that time, the light receiving devices that receive the light, as well as the strength of the light they receive, is determined by the manner in which that light passes through the LCDs. The letter in question is delineated by the light receiving devices that receive the strongest light. This system is capable of 100% letter recognition even when slightly misshapen handwritten letters are input.

A fourth example is a development project for a facilities diagnosis system that employs a neural network system commenced by the Nippon Oil Co. [6], Ltd. in cooperation with CSK Research Institute. This project is attracting considerable attention as it is the first time research has been carried out on applying neural network systems to facilities diagnosis. Initially, the project will be aimed at developing a diagnosis system for pump facilities that employs vibration analysis. Nippon Oil operates a total of 1,500 pumps at its Negishi Oil Refinery in Yokohama, Kanagawa Prefecture alone, and must retain large numbers of experienced personnel to maintain these pumps. The company decided to apply neural network technology to pump facilities diagnosis operations with the ultimate goal of saving labor in mind.

1.11 The Future

Neural Networks will fascinate user-specific systems for education, information processing, and entertainment. "Alternative ralities", produced by comprehensive environments, are attractive in terms of their potential for systems control, education, and entertainment. This is not just a far-out research trend, but is something which is becoming an increasing part of our daily existence, as witnessed by the growing interest in comprehensive "entertainment centers" in each home. This "programming" would require feedback from the user in order to be effective but simple and "passive" sensors (e.g fingertip sensors, gloves, or wristbands to sense pulse, blood pressure, skin ionisation, and so on), could provide effective feedback into a neural control system.

This could be achieved, for example, with sensors that would detect pulse, blood pressure, skin ionisation, and other variables which the system could learn to correlate with a person's response state.

Neural networks, integrated with other artificial intelligence technologies, methods for direct culture of nervous tissue, and other exotic technologies such as genetic engineering, will allow us to develop radical and exotic life-forms whether man, machine, or hybrid.

Neural networks will allow us to explore new realms of human capabillity realms previously available only with extensive training and personal discipline. So a specific state of consiously induced neurophysiologically observable awareness is necessary in order to facilitate a man machine system interface.

1.12 Some Advantages of Neural Network

Neural networks and artificial intelligence based models have potential advantages over other models due to following reasons

- Networks start processing the data without any preconceived hypothesis. They start with random weight assignment to various input variables. Adjustments are made based on the difference between predicted and actual output. This allows for unbiased and better understanding of data. It also some times help in unraveling subtle relationship between various input variables and out put being studied.
- Neural networks can be retrained using additional input variables and number of individuals. Once trained they can be called on to predict in a new patient.
- There are several neural network models available to chose from in a particular problem.
- Once trained, they are very fast.
- Due to increase accuracy, results in cost saving.

1.12.1 Disadvantages

- no set rule for network selection.
- Needs experts in training the network and in-depth understanding of medical problem in hand.
- training is quite time consuming and tests the patience.

1.13 Summary

A neural network is a system of programs and data structure that approximate the operation of human brain . A neural network usually involves a large number of processors operating in parallel.

In this chapter we cover the development of neurocomputing, how to build a neural network and some benefits of a neural network.

CHAPTER TWO
2. STRUCTURE OF RECURRENT NEURAL NETWORK

2.1 Overview

Recurrent neural networks are an attractive tool for both practical applications and for the modeling of biological nerve nets, but their successful application require an understanding of their dynamical properties, in particular, their stability.

The present work provides an in-depth study of this challenging issue and contributes a number of new results that are also important for a broader class of recurrent systems containing nonlinear and even time-delayed feedback.

The approach is based on modern concepts of control theory, with an emphasis on techniques that have been developed for the analysis of feedback systems with parameter uncertainties during recent years. In addition to the analytic derivations, the author demonstrates how the derived criteria can be numerically evaluated with modern techniques for quadratic optimization. Some of the techniques are then illustrated for the example of using a fully recurrent network for learning the dynamics of a chaotic system.
The present monograph will offer the mathematically inclined reader an unusual, but powerful approach to the stability analysis of recurrent systems and acquaint him with many advanced concepts that he may find useful for his own research.

2.2 Recurrent Networks

Artificial neural networks consist of a number of artificial neurons that simulate the characteristics of biological neurons in terms of the relationship between the input and the output [3]. As Figure 1 illustrates, every neuron has a set of input connections pi, a bias b, and an output a. Every input link has an input value p and a weight w. The total input, n, is calculated by multiplying every input value by the corresponding link weight and summing the products. The output of the neuron can be viewed as function:

$$a = f(\sum wp+b).$$

Various activation functions can be used to determine when and how the neuron fires (produces output).

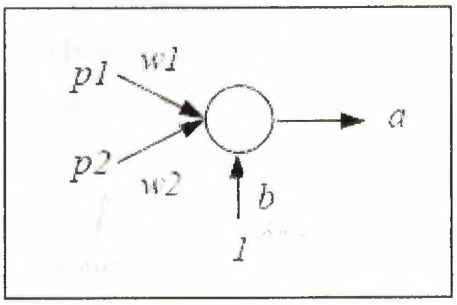


Figure 2.1. :A Single Artificial Neuron.

Neurons are grouped into layers with links from one layer to another, which forms a neural network (see Figure 2.1). If the links are always in one direction then the network is called a "feedforward neural network." On the other hand, if the network has backward connections then it is called a "recurrent neural network."

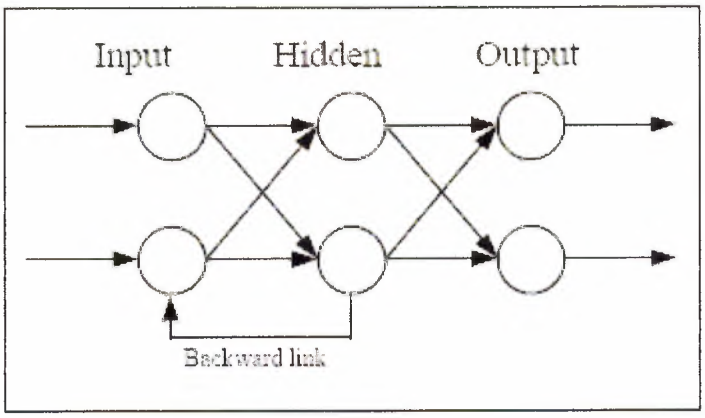


Figure 2.2. **:**Three-Layer Recurrent Neural Network.

2.3 Genetic Algorithms

Genetic algorithms are analogous to natural evolution [5]. They evolve populations consisting of individuals.

Every individual is defined by his genes. After creating an initial population, individuals with higher fitness are more likely to survive and produce offspring. When an individual is probabilistically selected based on his fitness, one of the following three operations is used: (1) crossover, (2) mutation, or (3) replication, Crossover requires two parents.

A part of the genetic data is exchanged between the two parents to produce two children. The advantage of this is to combine good features from both parents to produce a better individual. Mutation changes some of the genes to random values, which allows introducing new genetic information not currently present in the population. Some individuals may be replicated (copied) to preserve good combinations of genetic data.

2.4 Agent Structure

The agent has a number of internal variables that must remain positive (greater than zero) to keep the agent alive. When an input vector is passed to the agent, the agent processes it and decides which action is to be performed next.

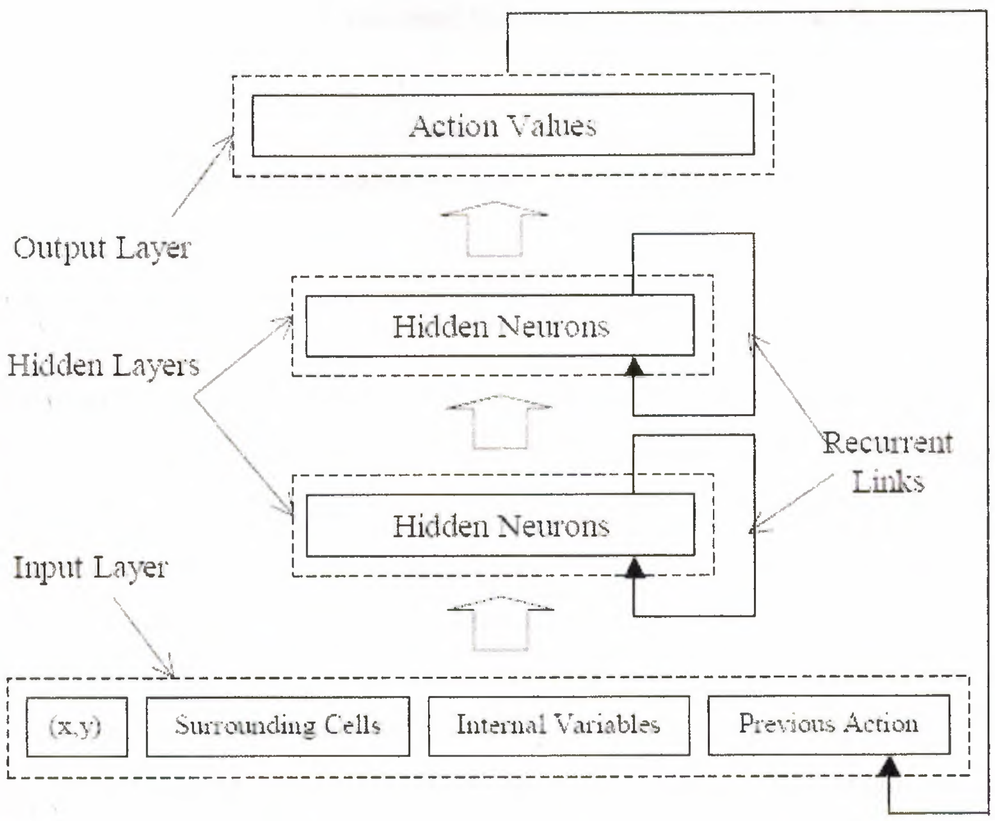


Figure 2.3. : Agent Structure.

The internal structure of the agent is a multi-layer RNN (see Figure 2.3). Every layer is fully connected with the next layer. Some of the nodes in the hidden layers have recurrent links.

The input layer accepts the following information as input:

- X and Y coordinate of the current cell.
- The type of the eight surrounding cells.
- The values of the internal variables.
- The previous action.

The input is then passed through a number of hidden layers with recurrent links.

Finally, the output layer contains nine outputs that represent a movement to one of the eight surrounding cells or staying at the current cell.

For example, if at time t an agent has two internal variables of value: (1) energy = 0.1 and (2) maintenance = 0.9, who is located in an energy cell (5, 5) of a 10x 10 worlds that are surrounded by empty cells, and assuming that the previous action was 9, the input would like as follow:

[0.5 0.5] [0.0 0.0 0.0 0.0 0.5 0.0 0.0 0.0 0.0] [0.1 0.9] [1.0]

Where the first set of values correspond to the coordinates of the agent divided by the world dimensions.

The second set of nine values corresponds to the types of the surrounding cells. A value of 0.0 indicates that the corresponding cell is an empty cell and a value of 0.5 corresponds to an energy cell (variable index divided by total number of variables). The third set of values consists of the current values of the internal variables. The last value is the previous action of the agent divided by 9, the number of possible actions. The output vector of the RNN would have nine values where the greatest value corresponds to the action to be performed. In this example, the action should be 5, which corresponds to remaining in the current cell and consuming the energy it provides. Thus, the output vector might contain the following values:

[-0.12 0.25 0.04 -0.78 0.75 -0.44 0.32 0.19 0.11]

2.5 Genome Structure

The genome completely defines an agent. It consists of two parts: (1) topology definition and (2) link weights. The topology definition consists of:

- H: the number of hidden layers
- N1: the number of nodes in the first hidden layer
- R1: the number of recurrent links in the first hidden layer
- NH: the number of nodes in the Hth hidden layer
- RH: the number of recurrent links in the first hidden layer.

The link weights are sets of two-dimensional arrays where each array Ai represents the weights of links from layer i to layer i+1.

In addition, the initial outputs of the recurrent neurons are also represented in the genome.

2.6 Recurrent Neural Network Prosperities

2.6.1 R.N.N Consist of Simple Constituents

A recurrent neural network is composed of formal neurons, which are the basic computational units of the network.

Each neuron computes a weighted sum of its inputs and performs an additional non-linear transformation on the result of this summation.Thus the neurons are relatively simple units and it is even possible to build them as hardware circuits.

Between each pair of two neurons there is a directed connection, through which the output of one neuron becomes the input of the other.

Supplied with the connection is a weight parameter, which controls the magnitude of the incoming signal in the weighted sum.

2.6.2 R.N.N Show complex collective behaviors

There are mainly two aspects, which makes it hard to analyse a recurrent neural network.

First it usually is supposed to have many neurons and with the number of neurons the number of weights grows quadratically. And second, every neuron contributes to the computation in the network through a non-linear function, which makes the system as a whole highly non-linear and requires sophisticated methods to obtain results about its collective behavior.

2.6.3 R.N.N Is Biologically Inspired

As the terminology 'Neural Network' indicates, much of the initial motivation for artificial neural network research came from drawing analogies to the nervous system [Heb49, MP43]. Loosely speaking the weighted summation performed by the formal neuron can be compared to the integration of incoming signals in a biological neuron, and the output after application of the non-linear function with the mean firing rate of the real cell.

It is an open discussion how fruitful this analogy is because obviously most of the rich biological structure of a real neuron is missed by such simple formal cell models.

On the other hand it has very often been conjectured that even the relatively primitive artifical models may provide substantial insight, which processes can potentially be responsible for the vast learning and computational skills of the real brain [DMM90].

Further, there is also an entire subfield of modelling the formation of structure and specialization of cells which are found in visual cortex, where the newest models incorporate also feedback.

Finally there are attempts to use more realistic models for the neurons including oscillators [BH96, TW95] and spiking neurons [Mas98]. Despite of that extensive work we are at present neither concerned with biological neurons or their models, nor do we aim at direct conclusions for biological plausibility of certain learning or computational processes from our work.

2.6.4 R.N.N Is a Theoretical Modeling Paradigm

Our subjects are theoretical aspects of (recurrent) neural computation and we regard (recurrent) neural networks mainly as a special paradigm for modeling some arbitrary process.

As recurrent networks have internal dynamics they are natural candidates for the implementation of transforms between dynamically changing input functions and corresponding output functions.

This property has often been stressed as generic to use them for identification and modeling of dynamical plants in a control theoretic framework A major advantage of this paradigm is the distributed way the computation is organized in the network, which makes it robust and fault tolerant and lets it easily deal with noisy input.

On the other hand we can not understand how the network performs a certain computation, because the network does not provide any explicit representation, neither of the problem nor of the problem data.

2.6.5 R.N.N Is Universal Approximates and Perform Hard Tasks

The usage of neural networks, whether recurrent or feed forward, grounds in their remarkable approximation capabilities. It can be shown that a recurrent network can approximate arbitrarily well any finite time trajectory of a dynamical system defined by a continuous vector field [FN93].

This is mainly the case because the vector field can be approximated by a feedforward network [Olu94], which then is embedded into a larger recurrent network.
Further it has been shown that a feedforward network can also approximate a given input-output behaviour of any non-linear operator defined on some compact subset of the space of all continuous or integrable functions [CC95].

It is an open question whether this last result can also be transfered to the recurrent network domain and what it would mean in that context.
In any case it is widely believed that recurrent networks can be adapted to implement operators in function spaces and it has been shown in experiments that they can perform complex tasks.
We will use as application domain a neural network which learns an input-output transformation implicitly given by the dynamics of a chaotic attractor.

2.6.6 R.N.N is hard to use and do not perform hard tasks

The majority of general approximation theorems does not yield constructive methods to efficiently build a concrete approximation.
The theorems in neural network theory are no exclusion of this rule, which means that they yield little or nothing about how to determine size and connectivity of a concrete network for a given task.
Therefore, in some miraculous way, the experienced designer has to make the right choice of parameters, which is a major drawback of this whole modeling approach. Further, adaptation and evaluation of recurrent models are in general very time and resource consuming which makes them even worse suited for systematic search in parameter space.

We believe that these problems and the additional difficulties to obtain analytical results are the reasons why only a few concrete applications for recurrent networks have been described so far.

2.6.7 R.N.N Is Incrementally Adapted

Once we have chosen a network architecture and the characteristics of the formal neurons, we have to adjust the weight parameters to implement a certain behavior of the network.

It is central to all network theory that this adjustment can be done incrementally with respect to an error functional which has to express the desired behavior of the network [Pea95, Sch93]. Because of the simple structure of the neuron with differentiable non-linear transfer functions the chain rule can be used to evaluate the sensitivity of the output of the network to small changes of the weights.

Then the weights can be changed to improve the performance of the network with respect to the given output error. In the classical case this is the well known back propagation algorithm, which in its general form can be used also for recurrent networks [Pea95, Pea89]. But it has also been stressed that standard back propagation is only a special member of the more general class of gradient based methods [O'R96].

2.6.8 R.N.N is hard to adapt

Though it is conceptually clear how to adapt a network it is practically hard to do, especially in continuous time.

For each time step there are at least O(n3) operations, where n is the number of nodes in the network [Sch92].

There have to be chosen a learning rate, time constants for simulating the network itself, and for the corresponding sensitivity network. If the adaptation is done "on-line" at each time-step a bad choice of the learning rate can easily cause the network to diverge.

Behind this last point there is a serious theoretical problem: on-line learning couples weight and network dynamics, which become non-autonomous and then depend on time in some very complicated way.

Such networks can generally be described as interval bounded systems, where only Upper and lower bounds for the non-autonomous parameters, here the adapting weights, are specified.

2.6.9 R.N.N map input values into output values

Even if we apply constant input to a given recurrent network (which is not adapted) and assume that all trajectories approach some attractor, then in general the same input can yield different outputs (attractors) for different initial conditions.

Such a network does not define a proper mapping from inputs to attractors without resetting the initial values of all neurons every time a new input is applied [Pea95, Hir89].

To assure independence of the output from the initial conditions the network needs to have a unique equilibrium, which attracts all trajectories for all initial states. The corresponding mappings have attracted much interest and a number of state space stability conditions to obtain them have been derived.

2.6.10 R.N.N map input functions into output functions

A mapping from static input to static output can in principle as well be approximated by a feed forward network as by a recurrent one.

However, if time-varying input is supplied a recurrent network can use its internal dynamic structure to draw on the characteristics of the time-development of the input.

With time-varying input the output necessarily becomes time-varying as well.

Then, to receive finite output, it becomes important to assure that the input can not be infinitely be amplified by the network. In mathematical terms that mean that the function norm of the output function must be bounded by a finite multiple of the function norm of the input function.

More intuitively this means that the network may not develop internally driven activations like oscillations or chaotic motions, which clearly would disturb the functioning of the network as input-output device.

That networks are capable of oscillating and chaotic behavior has been shown in different settings [Ces94, SCS88] and can even be desirable in another context [ROT98, Fai98].

2.6.11 R.N.N can be regarded as input-output operators

The last remarks already point out the direction we take for the analysis of recurrent network behavior.

We are interested in a network's properties when time-dependent input functions are applied, which is the typical case if they are used for instance as control device or for the identification of some unknown system. From a general perspective then the network maps an input function into an output function, which can be formalized as an operation between the input function space and the output function space.

The network implements an operator between these spaces given by a non-linear feedback system.

2.7 Main Goals and Structure of the Work

The main focus of our work is a stability analysis of the input-output behavior of recurrent networks, because stability is a necessary prerequisite for wider application of recurrent networks as components in larger systems.

We believe that our work can contribute towards systems which can use the rich dynamical capabilities of recurrent networks to process time-varying inputs. In particular we derive

- A unified theoretical framework for time-invariant and time-varying networks with and without delays.
- Means to evaluate the networks input-output gain net.
- Efficient procedures to compute stability bounds.
- A new approach to stability under on-line adaptation.
- Generalizations of known state space results.

To achieve these results we need to introduce and partially extend a number of methods from non-linear feedback system theory.

The theoretical effort for this is reasonably high but results in a conceptually simple and clear framework from which it is straightforward to derive results for different types of networks.

Input-output stability of recurrent neural networks and computation of the corresponding gains has – up to our knowledge – not yet been addressed in the recurrent neural network research1.

We restrict ourselves to continuous systems because we aim to provide theoretical support for the application of recurrent networks for modeling and control of real continuous plants. There has only been one approach to treat a neural network as feedback circuit in [GC93], but with no impact to compute gains.

The key idea is to apply the decomposition principle introduced above for the evaluation of the behavior of interconnected systems to the recurrent network itself.

This is possible because a recurrent network can, due to its special structure, be decomposed into two subsystems, a linear feed forward system and a diagonal non-linear feedback.

This constellation is ideal for application of general theorems from feedback system theory and where we derive extensions of the known theory we especially indicate this.

The simple inner structure of recurrent networks allows modifying and often simplifying the general theorems to obtain results specialized to the recurrent network case.

We reserve the term "Criterion" to such results, which are all new in the field of neural networks.

Some of the resulting conditions have also been obtained by classical state space methods, but then under more severe restrictions.

Structuring this theses we can distinguish mainly three parts: first an introductory part, second the theoretical results on stability, and, third, computation and application.

The first part consists of three chapters in which we introduce and develop our approach and provide the necessary mathematical means.

2.8 Induction of Recurrent Neural Networks

2.8.1 Induction

Applying evolutionary computations to the training of neural networks has occurred to many researchers. Studies that use genetic algorithms to evolve connectionist networks generally avoid total network induction, i.e., inducing both the topology and parametric values of a network, favoring instead simple parametric learning.

Genetic algorithm studies that do induce network topology allow only limited structural changes.

In addition, these studies rely on generic forms of crossover to manipulate network structure, a process which this paper argues actually inhibits network evolution.

Evolutionary programming, an alternative evolutionary optimization technique, embodies principles that better respect the complexities of evolving neural networks. We present GNARL, an evolutionary program that induces both the topology and parametric values for recurrent neural networks.

GNARL is demonstrated on an interesting control task.

2.8.2 Evolving Networks with Genetic Algorithms

Genetic algorithms create new individuals by recombining the representational components of two population members, usually represented as bit strings which are interpreted as the representation to be evaluated.

They therefore rely on two distinct representational spaces, one in which the search is performed, called the search representation, and one in which candidate solutions are evaluated, called the evaluated representation.

An interpretation function maps between elements of these distinct representational spaces. For instance, when the task is to evolve connectionist networks, the interpretation function maps individuals from the chosen search representation into the evaluated connectionist network.

Clearly, the choice of interpretation function introduces a strong bias into the search, typically by excluding many potentially interesting and useful networks.

The benefits of having such a dual representation hinge on supplying an interpretation function that makes crossover an appropriate evolutionary operator for the task.

Otherwise, the need to translate between dual representations is an unnecessary complication.

Characterizing tasks for which crossover is beneficial is an open question. Crossover has been hypothesized to be most effective in tasks where the fitness of a member of the population is reasonably correlated with the expected ability of its representational components. Environments where this is not true are called deceptive. Deceptive problems are generally more difficult for a genetic algorithm to solve.

Often, the interpretation function is specially crafted to reduce or remove the deceptiveness of a task. The central question for evolving connectionist networks is then does a suitable interpretation function exist that adequately compensates for the deception inherent in the task of evolving connectionist networks so that crossover is a good operator. There are three forms of deception when using crossover to evolve connectionist networks.

The first involves identical networks, i.e., ones that share both a common topology and common weights when evaluated. Because the interpretation function may be many to-one, identical networks need not have the same search representation.

In such a situation, crossover will tend to create offspring that contain repeated components, thus losing the computational ability of some of the parents' hidden units. Consequently, the resulting networks will tend to perform worse than their parents since they are not in possession of key computational components for the task.

This has been called the competing conventions problem, The second form of deception involves two networks with identical topologies but different weights.

It is well known that for a given task, a single connectionist topology affords multiple solutions for a task, each implemented by a unique distributed representation spread across the hidden units. The computational role each node plays in the overall representation of the task solution is determined purely by the presence and strengths of its interconnections.

As a result, there need be no correlation between distinct distributed representations over particular network architecture for a given task.

This greatly reduces the possibility that an arbitrary crossover operation between distinct distributed representations will construct viable offspring regardless of the interpretation function used.

Finally, deception can occur when the parents differ topologically. The types of distributed representations that can develop in a network vary widely with the number of hidden units and the network's connectivity.

Thus, the distributed representations of topologically distinct networks have a greater chance of being incompatible parents.

In short, for crossover to be an appropriate operator for evolving networks, the interpretation function must somehow compensate for the various types of deception described above.

The discussion above suggests that for general network induction the complexity of an appropriate interpretation function will more than rival the complexity of the original learning problem.

Thus, the prospect of evolving connectionist networks with crossover goes against the current theory associated with both genetic algorithms and connectionist networks, and better results should be expected with reproduction heuristics that respect the uniqueness of the independently developing distributed representations.

This point has been tacitly validated in the genetic algorithms literature by a trend towards a reduced reliance on binary representations when evolving networks. However, the juxtaposition of distributed representations via crossover is still commonplace.

For complete network induction without unnecessarily restricting the search space of architectures, alternative operators that respect the idiosyncrasies of the developing architectures are required.

2.8.3 Using an Evolutionary Program to Evolve Connectionist Networks

In evolutionary programming (EP), there is no dual representation scheme as in Genetic algorithms.

The actual representation evaluated by the fitness function is manipulated directly. Thus no interpretation function is required. Further, evolutionary programs generally avoid recombination of any form, choosing instead to manipulate structures by representation specific mutation operators.

EP mutation operators are more complex than the weak form of mutation used in genetic algorithms and often are designed to respect the idiosyncrasies of the chosen representation. This provides a significant advantage for EP methods since EP operators are supplied more task specific knowledge than typically used in standard genetic operators.

By using informed representation specific operators and avoiding recombination, EP is a much better paradigm for evolving the weights and topology of recurrent connectionist networks.
These qualities allow the developing distributed representations that are unique to each network in the population to be manipulated in a manner consistent with their nature.
Previous EP studies that evolve connectionist networks, typically fall into the same traps of architecture restriction as genetic algorithms.
In the next section, we describe GNARL, an evolutionary program that performs complete network induction.

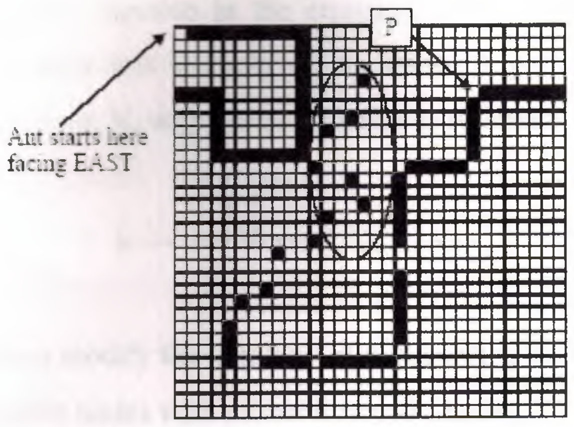
2.9 The GNARL Algorithm

The GNARL (Generalized Acquisition of Recurrent Links) algorithm begins with an initial population of n randomly generated networks.
The number of input nodes (num-in) and number of output nodes (num-out) are fixed for a given task; the number of hidden nodes as well as the connections among them are free to vary self-links as well as general recurrent loops are permitted.
In each generation of search, the networks are ranked by a user-supplied fitness function

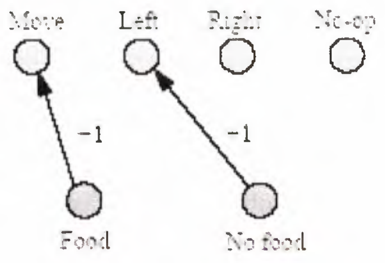
$$f: N \rightarrow R$$

The flexibility of evolutionary methods allows a number of fitness metrics to be used.
After ranking, the best n/2 individuals are saved and allowed to reproduce with mutations each generation.
Reproduction entails modifying both the weights and topology of each parent network N.
First, the temperature of a network T(N) is calculated:

$$T(N) = U(0,1)\left(1 - \frac{f(N)}{f_{max}}\right)$$



(a)



(b)

Figure 2.4. :The Tracker Task. (a)The trail is connected initially (b)The semantics of the I/O units for the ant network.

Where fmax (provided by the user) is the maximum possible fitness for a given task and U (0, 1) is a uniform random variable over the interval (0, 1).

This measure of N's performance is used to anneal both the structural and parametric similarity between parent and offspring according to the network's proximity to correct performance.

Networks with a high temperature, and thus poor performance, are mutated more severely while those with a low temperature, and thus nearer to a solution, are mutated only slightly. This measure encourages a coarse-grained search initially and a finer-grained search as a network approaches a solution.

The uniform random variable in the equation allows even a network that is far from optimal to be mutated less severely on occasion. A parametric mutation perturbs each weight, w, in a network, N, with Gaussian noise as follows:

$$w \leftarrow w + \mathrm{Normal}(0, T(N)), \quad \forall w \in N$$

Structural mutations modify the topology of N according to the following:

- add k1 hidden nodes with probability padd-node
- delete k2 hidden nodes with probability pdelete-node
- add k3 links with probability padd-link
- delete k4 links with probability pdelete-link

Where each k i is selected uniformly from a user-defined range, again annealed by T (N). When a node is added, it is initialized without connections; when a node is deleted, all its Incident links are removed.
All new links are initialized to 0

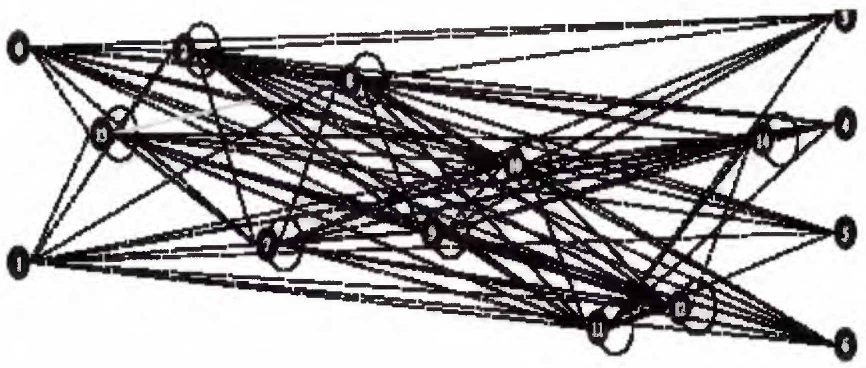


Figure 2.5. : The Tracker Task, first run. Network induced by GNARL after 2090 generations.

Forward links are dashed; bidirectional links & loops are solid. The light gray connection between nodes 8 and 13 is the sole backline. This network clears the trail in 319 epochs.

2.11 Discussion and Conclusions

In the original study, Jefferson ET al.12 use a genetic algorithm, and hence crossover, to evolve only the parameters of a recurrent neural network with five hidden units for the Tracker task. They report that 1,123,942 networks were generated to evolve a network of similar ability to the networks evolved by GNARL in the two runs.

This is 10.74 and 14.07 times the number of recurrent individual networks generated respectively in the two

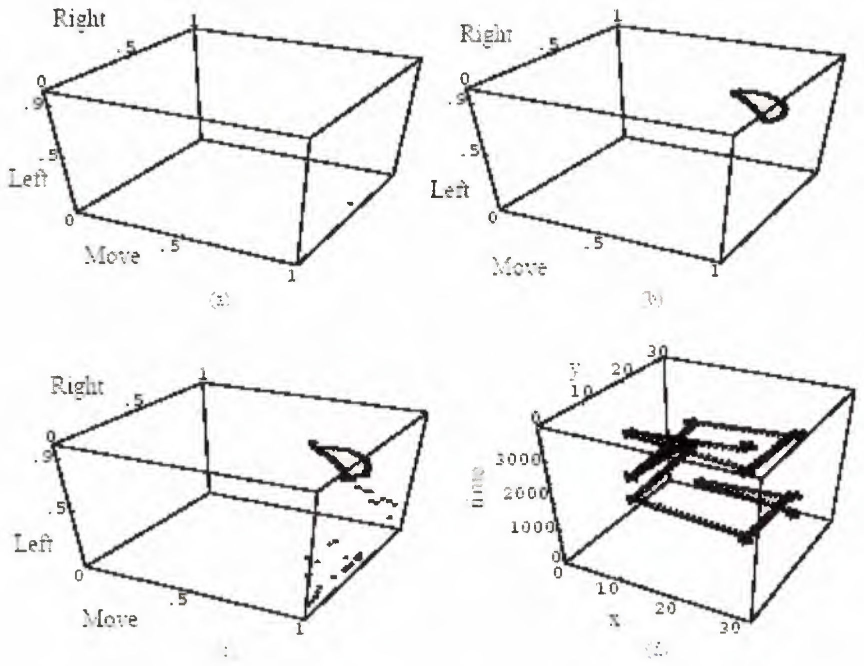


Figure 2.6 : Limit behavior of the network of the second run. Graphs show the state of the output units Move, Right, Left.

(a) Fixed point attractor those results for sequence of 500 "food" signals.

(b) Limit cycle attractor that results when a sequence of 500 "no food" signals is given to network.

(c) All states visited while traversing the trail.

(d) The x position of the ant over time when run on an empty grid.

runs of GNARL. Thus GNARL constructed an order of magnitude fewer recurrent networks in both runs to solve the problem. The fact that GNARL was also manipulating the architecture of the recurrent network while the architecture was static in Jefferson ET al.12 makes this difference especially significant. While it is tempting to use the above results to validate our claim of the inappropriateness of crossover for evolving networks, there may be other explanations.

For instance, the synergy of manipulating both the parameters and topology of the network may also contribute to the quicker learning. Consider that fixed network architecture, if chosen poorly, can inhibit the prompt evolution of a solution. Next, consider that in GNARL, architectures are indirectly selected for how quickly they improve over the task.

This encourages the identification of architectures that are conducive to manipulation by the selected mutation operators, potentially leading to quicker learning.

However, by the arguments we feel that the mismatch between crossover and the network representation should be the most significant reason for the speed-up shown by GNARL.

CHAPTER THREE

3. TRAINING RECURRENT NETWORKS FOR CONTROL

3.1 Overview

Neural networks can be classified into recurrent and nonrecurrent categories. Nonrecurrent (feedforward) networks have no feedback elements; the output is calculated directly from the input through feedforward connections. In recurrent networks the output depends not only on the current input to the network, but also on the current or previous outputs or states of the network. For this reason, recurrent networks are more powerful than nonrecurrent networks and have important uses in control and signal processing applications. This chapter introduces the Layered Digital Recurrent Network (LDRN), develops a general training algorithm for this network, and demonstrates the application of the LDRN to problems in controls and signal processing. we present the notation necessary to represent the LDRN.

The concepts underlying the backpropagation-through-time and forward perturbation algorithms are presented in a unified framework and are demonstrated for a simple, single-loop recurrent network. And we describe a general forward perturbation algorithm for computing training gradients for the LDRN. Two application sections follow the discussion of dynamic backpropagation: neurocontrol and nonlinear filtering.

This chapter demonstrate the implementation of the general dynamic backpropagation algorithm. The control part applies a neurocontrol architecture to the automatic equalization of an acoustic transmitter. The nonlinear filtering part demonstrates the application of a recurrent filtering network to a noise-cancellation application.

3.2 Layered Feed Forward Network

Figure 3.1 is an example of a layered feedforward network (two layers in this case). (See Demuth et al. [1998] for a full description of the notation used here.) The input vector to the network is represented by p1 , which has R1 elements. The superscript represents the input number, since it is possible to have more than one input vector. The input is connected to Layer 1 through the input weight IW1 1 , where the first superscript represents the layer number and the second superscript represents the input number. The bias for the first layer is represented by . The net input to Layer 1 is denoted by b1 , and is computed as n1 , and is computed as

$$n^1 = IW^{1,1} p^1 + b^1 \quad (1)$$

The output of Layer 1, a1 is computed by passing the net input through a transfer function, according to a1= f1. n1.

The output has S1 elements. The output of the first layer is input to the second layer through the layer weight LW2 1.

The overall output of the network is labeled y. This is typically chosen to be the output of the last layer in the network, as it is in Figure 3.1, although it could be the output of any layer in the network.

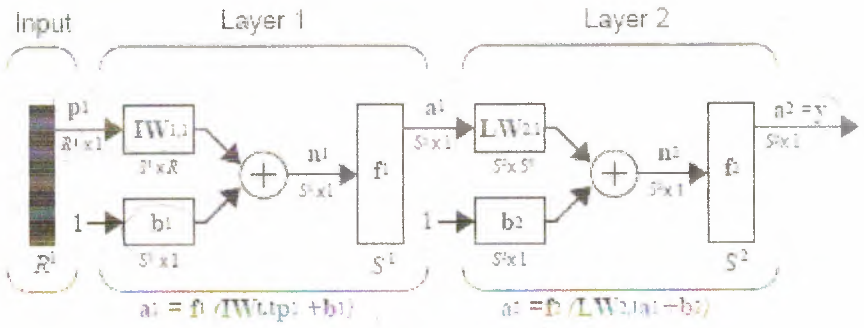


Figure 3.1. : Example of a Layered Feedforward Network

Each layer in the LFFN is made up of 1) a set of weight matrices that come into that layer (which may connect from other layers or from external inputs), 2) a bias vector, 3) a summing junction, and 4) a transfer function. (In the LDRN, a set of tapped delay lines may also be included in a layer, as we will see later.) In the example given in Figure 3.1, there is only one weight matrix associated with each layer, but it is possible to have weight matrices that are connected from several different input vectors and layer outputs. This will become clear when we introduce the LDRN network. Also, the example in Figure3.1 has only two layers; our general LFFN can have an arbitrary number of layers. The layers do not have to be connected in sequence from Layer 1 to Layer M. For example, Layer 1 could be connected to both Layer 3 and Layer 4, by weights LW3 1, andLW4 1, respectively. Although the layers do not have to be connected in a linear sequence by layer number, it must be possible to compute the output of the network by a simple sequence of calculations. There cannot be any feedback loops in the network. The order in which the individual layer outputs must be computed in order to obtain the correct network output is called the simulation order.

3.3 Layered Digital Recurrent Network

We now introduce a class of recurrent networks that are based on the LFFN. The LFFN is a static network, in the sense that the network output can be computed directly from the network input, without the knowledge of initial network states. A Layered Digital Recurrent Network (LDRN) can contain feedback loops and time delays. The network response is a function of network inputs, as well as initial network states. The components of the LDRN are the same as those of the LFFN, with the addition of the tapped delay line (TDL), which is shown in Figure 3.2. The output of the TDL is a vector containing current and previous values of the TDL input. In Figure 3.2 we show two abbreviated representations for the TDL. In the case on the left, the undelayed value of the input variable is included in the output vector. In the case on the right, only delayed values of the input are included in the output.

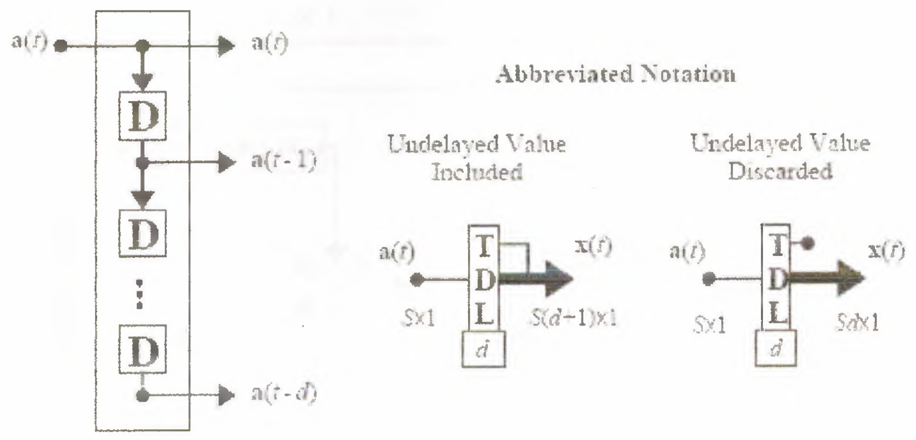


Figure 3.2. : Is an example of an LDRN.

Like the LFFN, the LDRN is made up of layers. In addition to the weight matrices, bias, summing junction, and transfer function, which make up the layers of the LFFN, the layers of the LDRN also include any tapped delay lines that appear at the input of a weight matrix. (Any weight matrix in an LDRN can be proceeded by a tapped delay line.) For example, Layer 1 of Figure 3.3 contains the weight LW1 2.. and the TDL at its input. Note that all of the layer outputs and net inputs in the LDRN are explicit functions of time. The output of the TDL in Figure 3.3 is labeled a-2. 2 (t) . This indicates that it is a composite vector made up of delayed values of the output of Subnet 2 (indicated by the second superscript) and is an input to Subnet 2 (indicated by the first superscript). (A subnet is a series of layers which have no internal tapped delay lines. The number of the subnet is the same as the number of its output layer. These concepts are defined more carefully in a later section.) These TDL outputs are important variables in our training algorithm for the LDRN.

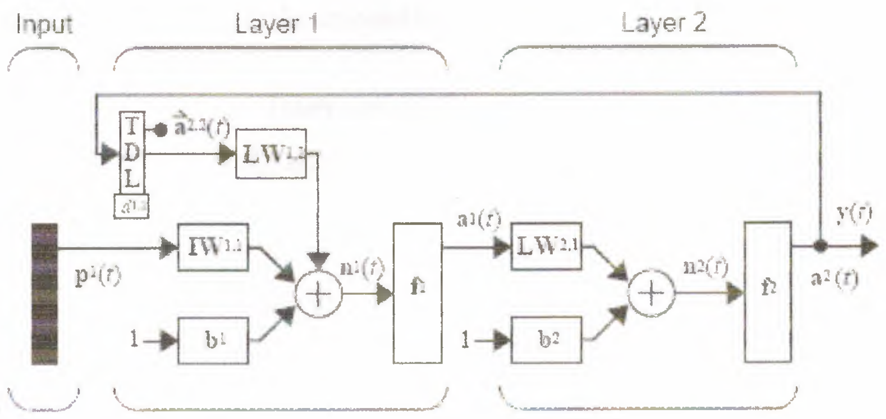


Figure 3.3. : Layered Digital Recurrent Network Example

In the LDRN, feedback is added to an LFFN. Therefore, unlike the LFFN, the output of the network is a function not only of the weights, biases, and network input, but also of the outputs of some of the network layers at previous points in time. For this reason, it is not a simple matter to calculate the gradient of the network output with respect to the weights and biases (which is needed to train the network). This is because the weights and biases have two different effects on the network output. The first is the direct effect, which can be calculated using the standard backpropagation algorithm [Hagan et al., 1996]. The second is an indirect effect, since some of the inputs to the network, such as a-1.2(t) are also functions of the weights and biases. In the next section we briefly describe the gradient calculations for the LFFN and show how they must be modified for the LDRN. The main development of the next two sections is a general gradient calculation for arbitrary LDRN's.

3.4 Principles of Dynamic Learning

Consider again the multilayer network of Figure 3.1. The basic simulation equation of such a network is

$$a^k = f^k\left(\sum_i IW^{k,i} p^i - \sum_j LW^{k,j} a^j + b^k\right), \qquad (2)$$

where k is incremented through the simulation order.

The task of the network is to learn associations between a specified set of input/output pairs: {(p1, t1), (p2, t2), ... , (pQ, tQ)}. The performance index for the network is

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{y}_q)^T (\mathbf{t}_q - \mathbf{y}_q) = \sum_{q=1}^{Q} \mathbf{e}_q^T \mathbf{e}_q \tag{3}$$

where yq is the output , pq of the network when the qth input, , is presented, and x is a vector containing all of the weights and biases in the network. (Later we use to represent the weights and biases in Layer i.) The network should learn the vector that minimizes .

For the standard backpropagation algorithm [Hagan et al., 1996] we use a steepest descent learning rule. The performance index is approximated by:

$$\hat{F} = \mathbf{e}_q^T \mathbf{e}_q . \tag{4}$$

where the total sum of squares is replaced by the squared errors for a single input/ output pair. The approximate steepest (gradient) descent algorithm is then:

$$\Delta w_{i,j}^{k,l} = -\alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{k,l}}, \quad \Delta b_i^k = -\alpha \frac{\partial \hat{F}}{\partial b_i^k} \tag{5}$$

$$s_i^k \equiv \frac{\partial \hat{F}}{\partial n_i^k} \tag{6}$$

as the sensitivity of the performance index to changes in the net input of unit i in layer k. Using the chain rule, we can show that

$$\frac{\partial \hat{F}}{\partial i w_{i,j}^{m,l}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial i w_{i,j}^{m,l}} = s_i^m p_j^l, \quad \frac{\partial \hat{F}}{\partial l w_{i,j}^{m,l}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial l w_{i,j}^{m,l}} = s_i^m a_j^l,$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m} = s_i^m \tag{7}$$

It can also be shown that the sensitivities satisfy the following recurrence relation, in which m is incremented through the backpropagation order, which is the reverse of the simulation order:

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m) \sum_i (\mathbf{LW}^{i,m})^T \mathbf{s}^i \tag{8}$$

where

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(n_{S^m}^m) \end{bmatrix} \tag{9}$$

and

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m} \tag{10}$$

This recurrence relation is initialized at the output layer:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}_q - \mathbf{y}_q) \tag{11}$$

The overall learning algorithm now proceeds as follows: first, propagate the input forward using Eq. (2); next, propagate the sensitivities back using Eq. (11)and Eq. (8); and finally, update the weights and biases using Eq. (5) and Eq. (7).Now consider an LDRN, such as the one shown in Figure 3. Suppose that we use the same gradient descent algorithm, Eq. (5), that is used in the standard backpropagation algorithm. The problem in this case is that when we try to find the equivalent of Eq. (7) we note that the weights and biases have two different effects on the network output. The first is the direct effect, which is accounted for by Eq. (7). The second is an indirect effect, since some of the inputs to the network, such as a-1.2(t) , are also functions of the weights and biases. To account for this indirect effect we must use dynamic backpropagation.

To illustrate dynamic backpropagation [Yang et al., 1993, Yang, 1994], consider Figure 3.4, which is a simple recurrent network. It consists of an LFFN with a single feedback loop added from the output of the network, which is connected to the input of the network through a single delay. In this figure the vector x represents all of the network parameters (weights and biases) and the vector a(t) represents the output of the LFFN at time step t.

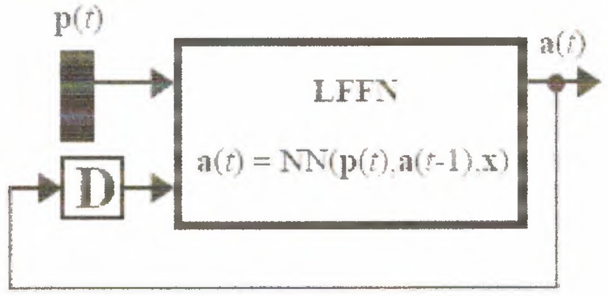


Figure 3. 4. : Simple Recurrent Network

Now suppose that we want to minimize

$$F(\mathbf{x}) = \sum_{t=1}^{Q} (\mathbf{t}(t) - \mathbf{a}(t))^T (\mathbf{t}(t) - \mathbf{a}(t)) \tag{12}$$

In order to use gradient descent, we need to find the gradient of F with respect to the network parameters. There are two different approaches to this problem. They both use the chain rule, but are implemented in different ways:

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}}\right]^T \times \frac{\partial^e F}{\partial \mathbf{a}(t)} \tag{13}$$

or

$$\frac{\partial F}{\partial \mathbf{x}} = \sum_{t=1}^{Q} \left[\frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}}\right]^T \times \frac{\partial F}{\partial \mathbf{a}(t)} \tag{14}$$

where the superscript e indicates an explicit derivative, not accounting for indirect effects through time. The explicit derivatives can be obtained with the standard backpropagation algorithm, as in Eq. (8). To find the complete derivatives that are required in Eq. (13) and Eq. (14), we need the additional equations:

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}} = \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{x}} - \frac{\partial^e \mathbf{a}(t)}{\partial \mathbf{a}(t-1)} \times \frac{\partial \mathbf{a}(t-1)}{\partial \mathbf{x}} \tag{15}$$

and

$$\frac{\partial F}{\partial \mathbf{a}(t)} = \frac{\partial^e F}{\partial \mathbf{a}(t)} + \frac{\partial^e \mathbf{a}(t-1)}{\partial \mathbf{a}(t)} \times \frac{\partial F}{\partial \mathbf{a}(t+1)} \tag{16}$$

Eq. (13) and Eq. (15) make up the forward perturbation (FP) algorithm. Note that the key term is

$$\frac{\partial \mathbf{a}(t)}{\partial \mathbf{x}} \tag{17}$$

which must be propagated forward through time.Eq. (14) and Eq. (16) make up the backpropagation-through-time (BTT) algorithm. Here the key term is

$$\frac{\partial F}{\partial \mathbf{a}(t)} \quad (18)$$

which must be propagated backward through time. In general, the FP algorithm requires somewhat more computation than the BTT algorithm. However, the BTT algorithm cannot be implemented in real time, since the outputs must be computed for all time steps, and then the derivatives must be backpropagated back to the initial time point. The FP algorithm is well suited for real time implementation, since the derivatives can be calculated at each time step.

3.5 Dynamic Backprop for The LDRN

In this section, we generalize the FP algorithm, so that it can be applied to arbitrary LDRN's. This is followed by applications of the LDRN and dynamic backpropagation to problems in filtering and control.

3.5.1 Preliminaries

To explain this algorithm, we must create certain definitions related to the LDRN. We do that in the following paragraphs. First, as we stated earlier, a layer consists of a set of weights, associated tapped delay lines, a summing function, and a transfer function. The network has inputs that are connected to special weights, called input weights, and denoted by IWi-j , where. j denotes the number of the input vector that enters the weight, and denotes the number of the layer to which the weight is connected. The weights connecting one layer to another are called layer weights and are denoted by LWi-j, where j denotes the number of the layer coming into the weight and I denotes the number of the layer at the output of weight. In order to calculate the network response in stages, layer by layer, we need to proceed in the proper layer order, so that the necessary inputs at each layer will be available. This ordering of layers is called the simulation order. In order to backpropagate the derivatives for the gradient calculations, we must proceed in the opposite order, which is called the backpropagation order.

In order to simplify the description of the training algorithm, the LDRN is divided into subnets. A subnet is a section of a network that has no tapped delay lines, except at the subnet input. Every LDRN can be organized as a collection of subnets. We define the subnets by proceeding backwards from the last subnet to the first subnet. To locate the last subnet, start at the first layer in the backpropagation order and proceed through the backpropagation order until you find a layer containing delays, which becomes the first layer in the last subnet. The last subnet is then defined as containing all of the layers beginning at the layer containing delays and continuing through the simulation order to the first layer in he backpropagation order (or the last layer in the simulation order). This defines the last subnet. To find the preceding subnet, start with the next layer in the backpropagation order and proceed in the same way until you find the next layer with delays. This process continues until you reach the last layer in the backpropagation order, at which time the first subnet is defined. As with the layer simulation order, we can also define a subnet simulation order that starts at thefirst subnet and continues until the last subnet.

For example, the LDRN shown in Figure 3.5 has thee layers and two subnets. To simplify the algorithm, the subnet is denoted by the number of its output layer. For this network the simulation order is 1-2-3, the backpropagation order is 3-2-1 and the subnet simulation order is 1-3.

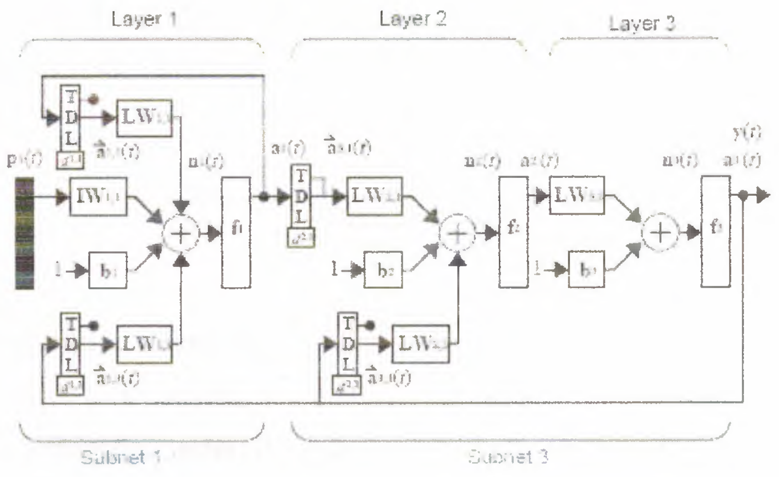


Figure 3.5. : Three-layer LDRN with Two Subnets

3.5.2 Explicit Derivatives

We want to generalize the forward perturbation (FP) algorithm of Eq. (13) and Eq. (15) so that it can be applied to an arbitrary LDRN. Notice that we have two terms on the right-hand side of Eq. (15). We have an explicit derivative of the performance with respect to the weights, which accounts for the direct effect of the weights on the performance and can be computed through the standard backpropagation algorithm, as in Eq. (8). We also have a second term, which accounts for the fact that the weights have a secondary effect through the previous network output. In the general LDRN, we may have many different feedback loops, and therefore there could be many different terms on the right of Eq. (15), and each of those terms would have a separate equation like Eq. (15) to update the total derivative through time. For our development of the FP algorithm, we have one term (and an additional equation) for every place where one subnet is input to another subnet. Recall that the subnet boundaries are determined by the locations of the tapped delay lines. Within a subnet, standard backpropagation, as in Eq. (8),can be used to propagate the explicit derivatives, but at the subnet boundaries an equation like Eq. (15) must be used to calculate the total derivative, which includes both direct and indirect effects. In this subsection we describe the computation of the explicit derivatives, and then in the following subsection we explain the total derivative computation.

A backpropagation process to calculate the explicit derivatives is needed for each subnet. These equations involve calculating the derivative of the subnet output with respect to each layer output in the subnet. The basic equation is

$$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^i(t)} = \sum_k \frac{\partial^e \mathbf{n}^k(t)}{\partial \mathbf{a}^i(t)} \times \frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{n}^k(t)} \qquad (19)$$

where ajz (t) represents a subnet output, ai(t) is the output of a layer in subnet jz, nk (t) is the net input of layer k, which has a connection from layer i. The index I is incremented through the backpropagation order. If we define

$$\mathbf{S}^{k,jz} \equiv \frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{n}^k(t)}, \text{ and note that } \frac{\partial^e \mathbf{n}^k(t)}{\partial \mathbf{a}^i(t)} = \mathbf{LW}^{k,i}, \tag{20}$$

then we can write Eq. (19) as

$$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^i(t)} = \sum_k \mathbf{LW}^{k,i} \times \mathbf{S}^{k,jz}. \tag{21}$$

This recursion, where i is incremented along the backpropagation order, begins at the last layer in the subnet:

$$\frac{\partial^e \mathbf{a}^{jz}(t)}{\partial \mathbf{a}^{jz}(t)} = \mathbf{I}, \tag{22}$$

where I is an identity matrix whose dimension is the size of layer jz.

3.5.3 Complete FP Algorithim for The LDRN

We are now ready to describe a generalized FP algorithm for the arbitrary LDRN. There are two parts to the FP algorithm: Eq. (13) and Eq. (15). Eq. (13) remains the same for the LDRN as for the simple network. Eq. (15), however, must be modified. For the general case we have one equation like Eq. (15) for each subnet output. Each of these equations has a term for the explicit derivative and one additional term for each subnet output. The complete FP algorithm for the LDRN network is given in the following flowchart. It contains three major sections. The first section computes the explicit (static) derivatives, as in Eq. (21), which are needed as part of the dynamic equations. The second section computes the dynamic derivatives of the subnet outputs with respect to the weights, as in Eq. (15). The final section computes the dynamic derivatives of performance with respect to the weights, as in Eq. (13).

3.6 Neurocontrol Application

In this section we illustrate the application of the LDRN and dynamic backpropagation to a control problem. We wish to control the output of an acoustic transmitter, whose schematic diagram is shown in Figure3. 6. Binary data is transmitted via acoustic signals from one pipeline location to another. Acoustic stress waves are imparted into the pipeline by the acoustic transmitter. The stress waves propagate through the pipeline to the acoustic receiver, which receives the transmitted signal. Tone bursts of different frequencies are used to represent either a 1 or a 0. The acoustic channel provided by the pipeline causes heavy distortion in the transmitted signal. There are often extraneous unwanted acoustic stress waves created by external sources, such as engines, geartrains, and pumps, that are imparted into the pipeline. The inherent channel distortion and the unwanted external noise can degrade the transmitted signal such that signal detection and interpretation at the receiver are unreliable or impossible. One method of enhancing communication performance is to equalize the effects of the transmission channel by adjusting the transmitter output so that the measured signal imparted to the pipeline at a short distance from the transmitter is the desired signal. Ideally, the feedback easurement of this signal is taken at the receiver. Of course, in practice, the feedback signal is measured near the transmitter. To alleviate the effects of unwanted disturbance noise, the transmitter can actively cancel the disturbance noise by way of destructive interference. The transmitter creates stress waves that are out of phase with, and of equal magnitude to, the undesired signals. In this example, a neural network controller is used as both a channel equalizer and an active noise canceller.

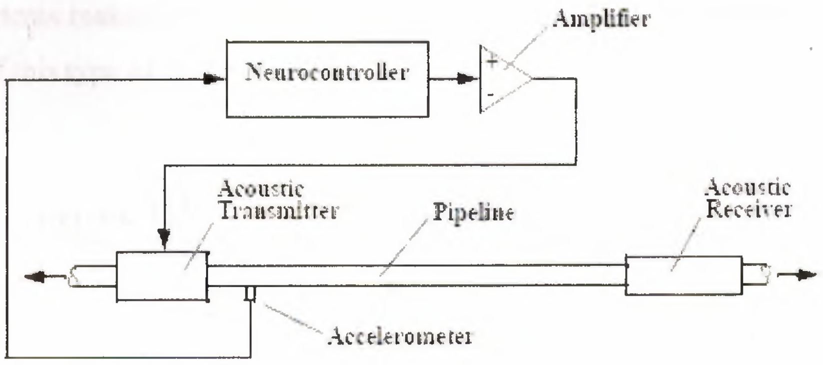


Figure 3.6. : Schematic of Acoustic Transmitter/Receiver System.

In addition to illustrating dynamic backpropagation on the neurocontroller for the acoustic transmitter, this section also demonstrates the effect of training with approximations to true dynamic derivatives. The evaluation is based on squared error performance and floating point operations. When approximations are used, the computational burden can be reduced, but the errors generally increase. It is a schematic of the control system. In this system, model reference adaptive control (MRAC) [Narendra, 1990] is applied to the control of the acoustic transmitter. The plant model is used only as a backpropagation path for the derivatives needed to adjust the controller weights; the plant model weights are not adjusted. The plant model is a 2-layer LDRN, with 10 input taps, 50 feedback taps, and 15 hidden neurons. The controller weights are adjusted such that the error ec(t), between a delayed reference input r(t) and the actual plant output c(t), is minimized. The controller structure consists of 40 input taps, 50 controller feedback taps, 75 plant output feedback taps, and 15 hidden neurons.

3.7 Recurrent Filter

This section provides a second example of the application of the LDRN and dynamic backpropagation. We use a multi-loop recurrent network to predict an experimental acoustic signal. The prediction of acoustic signals is crucial in active sound cancellation systems. Acoustic environments are often very complex, due to the complexity of typical sound sources and the presence of reflected sound waves. The dynamic nature of acoustical systems makes the use of recurrent filter structures of great interest for prediction and control of this type of system.

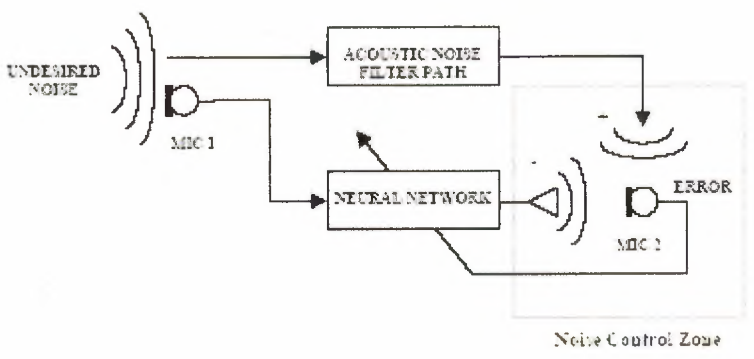


Figure 3.7. : Active Noise Control System

Figure 3.7 depicts a typical Active Noise Control (ANC) system. An acoustic noise source creates undesirable noise in a surrounding area. The goal of the active noise suppression system is to reduce the undesirable noise at a particular location by using a loudspeaker to produce "anti-noise" that attenuates the unwanted noise by destructive interference. In order for a system of this type to work effectively, it is critical that the ANC system be able to predict (and then cancel) unwanted sound in the noise control zone. In the first part of this section we develop the dynamic training equations for the ANC system. Then we present experimental results showing the prediction performance.

3.8 Supervised Learning Algorithims

The problem set-up of supervised learning in recurrent networks is similar to the case of feed-forward networks; a network is given a desired output for an input. An error function is defined and its gradient with respect to the weights are derived. However, the major difference is that the input and output are not static vectors but time sequences.

For example, in a recurrent network shown in Figure 3.8b, units 1 and 2 are output units, 3, 4, and 5 are hidden units, and 6 and 7 are input units. A small change of a connection weight, say w43 (shown as black dot) affects the output units, say unit 1, not only through the direct connection from unit 4 to 1 (thick black line), but also through indirect connections, through units 2, 3, and 5 (thick gray lines), and through infinitely many multi-step paths with multiple delays.

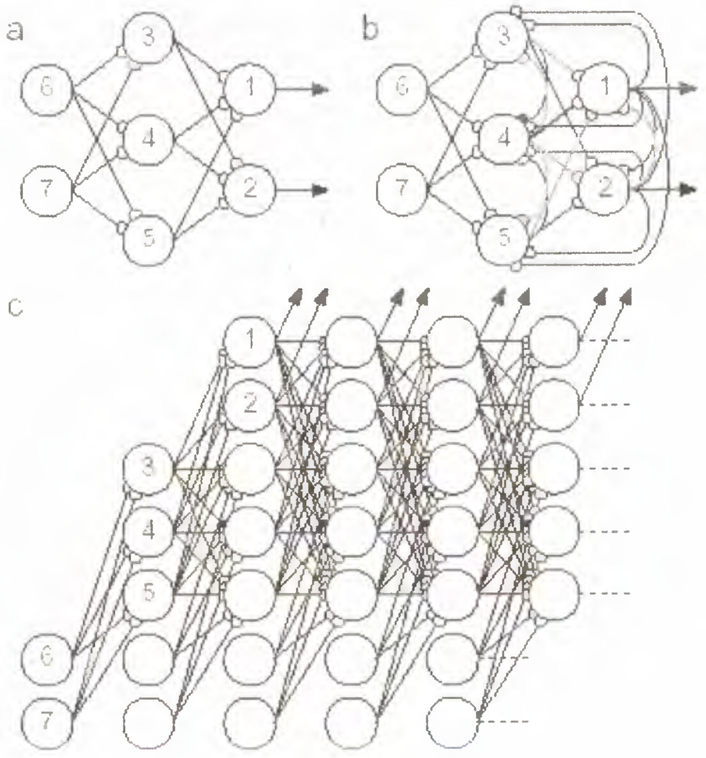


Figure 3.8.

It makes exact calculation of output error gradient rather complex. One simple strategy is to neglect all the indirect paths. In this case, although the network state evolves according to the recurrent network as in Figure 3.8b, a simple back-propagation algorithm is applied by regarding it as a feed-forward network as in Figure 3.8a. Such coarse approximation methods turned out to be effective in a series of work on language acquisition using simple recurrent networks that has recurrent connection between hidden units .

There are two basic ways to calculate the exact gradient of the output with respect to the weights: the forward methods and backward methods. The forward method estimates the effects of small change in a weight on the network state trajectory in a form of linear dynamic equation system.

This can be calculated concurrently with the network dynamic, thus useful for on-line learning. One drawback is the amount of computation for updating a set of dynamic equations for each weights.

The backward method estimates the causes for the output error backward in time.

In discrete time case, it is realized by 'unrolling' the multi-step evolution of the network state as a multi-layer feed-forward network, and applying the standard backpropagation .

In continuous-time case, it is done by running a set of 'adjoint system' backward in time. Although it requires asynchronous operation, the evolution and storage of the state trajectory first and error gradient calculation afterwards, the amount of computation is much less than in the forward methods.

3.9 Recurrent Backpropagation

So far we have been discussing the storage and retrieval of individual patterns. If we are interested in behavior in the real world, we normally do not work with one pattern but with a sequence of patterns.

Hopfield nets have been extended to include the possibility for storing and retrieving sequences (e.g. Kleinfeld, 1986), and for generating periodic motion (Kleinfeld and Somplinsky, 1989; Beer, 1992).

We will take look at recurrent backpropagation. A popular way to recognize (and sometimes reproduce) sequences has been to use partially recurrent networks and to employ context units. Figure 3.9 shows a number of different architectures. 3.9a is the suggestion by Elman (1990).

. The input units are separated into input units and context units, the context units hold a copy of the activation of the hidden units from the previous time step, the modifiable connections are all feedforward and can be trained by conventional backpropagation methods. Figure 3.9b shows Jordan's approach (Jordan, 1986).

It differs from 3.9a in that the output nodes are fed back into the context nodes and the context nodes are connected to themselves, thus providing a kind of short-term memory. The update rule is:

$$C_i(t+1) = \alpha C_i(t) + O_i(t)$$

where *Oi* are the output units and α is the strength of the self-connections. Such self-connecting units are sometimes call *decay units*, *leaky integrators*, or *capacitive units*. With *fixed* input, the network can be trained to generate a set of output sequences with different input patterns triggering different output sequences. With an *input sequence*, the network can be trained to recognize and distinguish different input sequences.

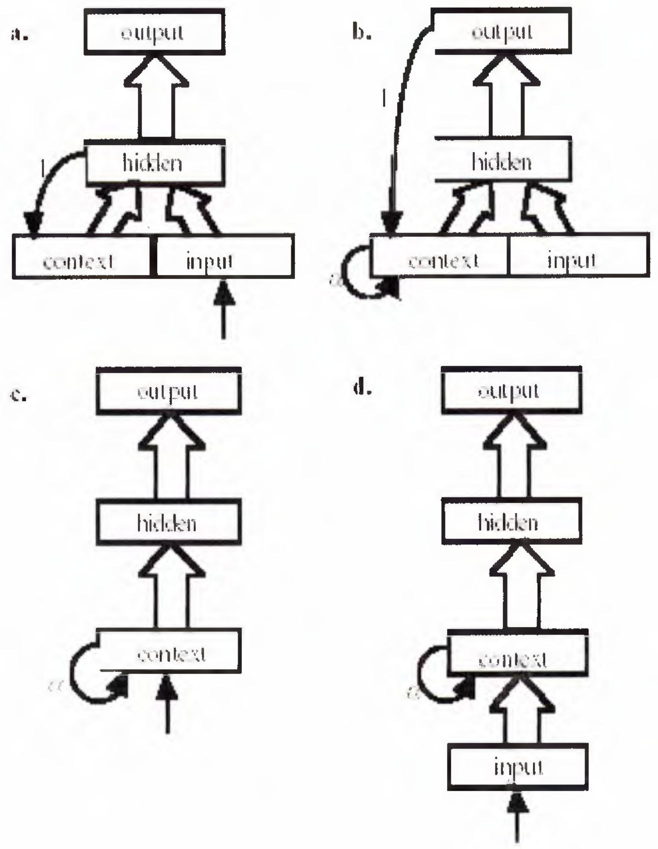


Figure 3.9. : Architectures with context units. Single arrows represent connections only from the i-th unit in the source layer to the i-th unit in the destination layer., whereas the wide arrows represent fully connected layers .

3.10 Summary

This chapter has discussed the training of recurrent neural networks for control and signal processing. When computing training gradients in recurrent networks, there are two different effects that we must account for. The first is a direct effect, which explains the immediate impact of a change in the weights on the output of the network at the current time. The second is an indirect effect, which accounts for the fact that some of the inputs to the network are previous network outputs, which are also functions of the weights. To account for this indirect effect we must use dynamic backpropagation.

This chapter has introduced the Layered Digital Recurrent Network (LDRN), which is a general class of recurrent network. A universal dynamic training algorithm for the LDRN was also developed. The LDRN was then applied to problems in control and signal processing. A number of practical issues must be addressed when applying dynamic training. Computational requirements for dynamic training can be much higher than those for static training, but static training is not as accurate and may not converge. The appropriate form of training to use (dynamic, static, or some combination) varies from problem to problem.

CHAPTER FOUR

4. APPLICATION OF RECURRENT NEURAL NETWORK FOR CONTROL

4.1 Overview

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.

4.2 Modelling of Neural Control System

Assume that control object is described by the following differential equation

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

Where a_i (i=1,n) and b_j (j=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 4.1 the structure of PID- neural controller is shown.

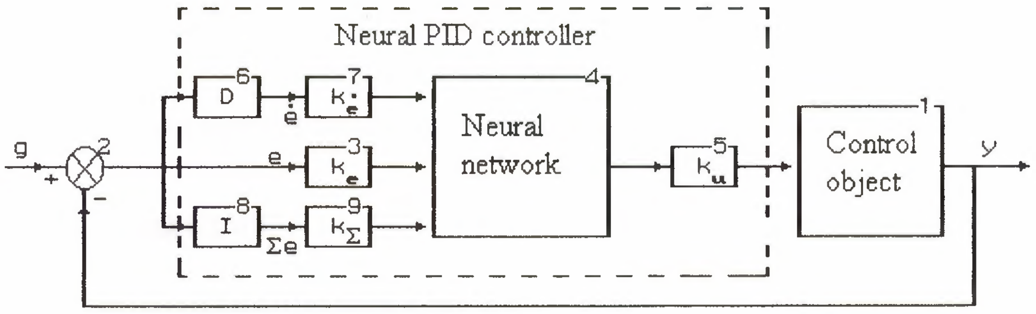


Figure 4.1. : Structure of Neural PID- controller.

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

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

Assume there is 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.

4.3 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 the models of control object are chosen by using following differential equations:

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

Where $a_0 = 0.072\ \text{min}^2$, $a_1=0.056$ min, $a_2=1$, $b_0=60\ ^{\circ}\text{C}/(\text{kgf}/\text{cm}^2)$;

Here y (t)- regulation parameter of object, u (t)- neural controller's output.

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

Where $a_0= 6.3\ \text{min}^2$, $a_1=11.2$ min, $a_2=1$, b0=5.1 $^{\circ}\text{C}/(\text{kgf}/\text{cm}^2)$, d=2.5 min is delay;

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

Where $a_0= 2.8\ \text{min}^3$, $a_1=3\ \text{min}^2$, $a_2=1$ min, $a_3=1$, $b_0=34\ ^{\circ}\text{C}/(\text{kgf}/\text{cm}^2)$;

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

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[$(\text{kgf}/\text{cm}^2)/^{\circ}\text{C}$] and differentiation time Td=0.15 min., then transient process in the control system oscillates with 18% of transient overshoot. Where settling time t=1.3-1.5 min, static error $\varepsilon_{st}(\infty) \approx \mathbf{0.15x(\infty)}$, 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 controlsystem 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.

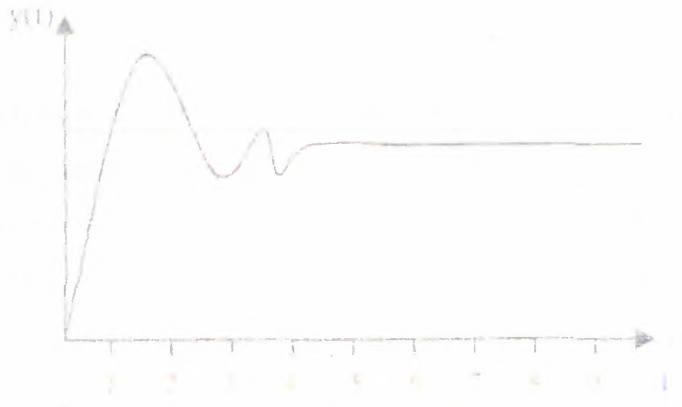


Figure 4.2. : (a) PD Controller

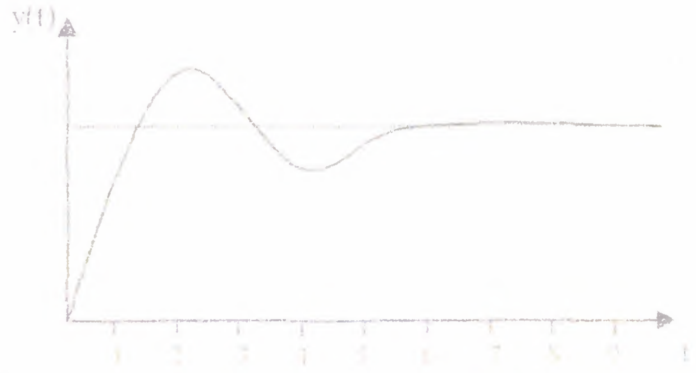


Figure 4.2. : (b) PI Controller

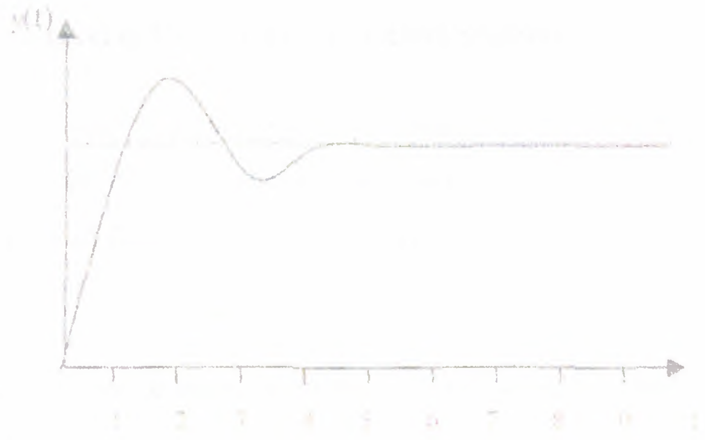


Figure 4.2. : (c) PID Controller

Figure 4.2.: 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)/^{o}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 ε_{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)/^{o}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 ε_{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.

4.4 Identification and Inverse Control of Dynamical Systems

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

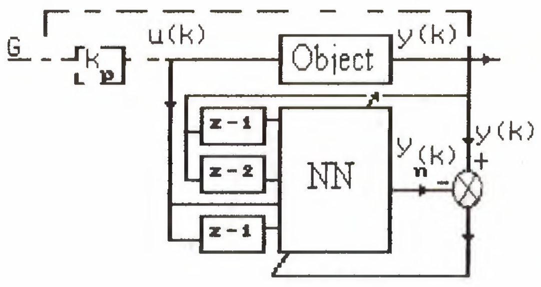
Also the direct and inverse identifications of control object (1) and development of inverse controller are performed.

In fig 4.3. :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 (fig.4.4) 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.



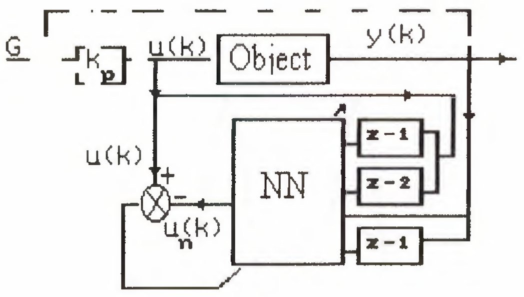


Figure 4.3. : Structure of direct identification **Figure 4.4.** : Structure of inverse Identification

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 4.5(a, b). During the identification the sinusoidal signal is given to the input of the system. In the figure the straight line shows the obect output (4.5a) and object input (4.5b) 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 .

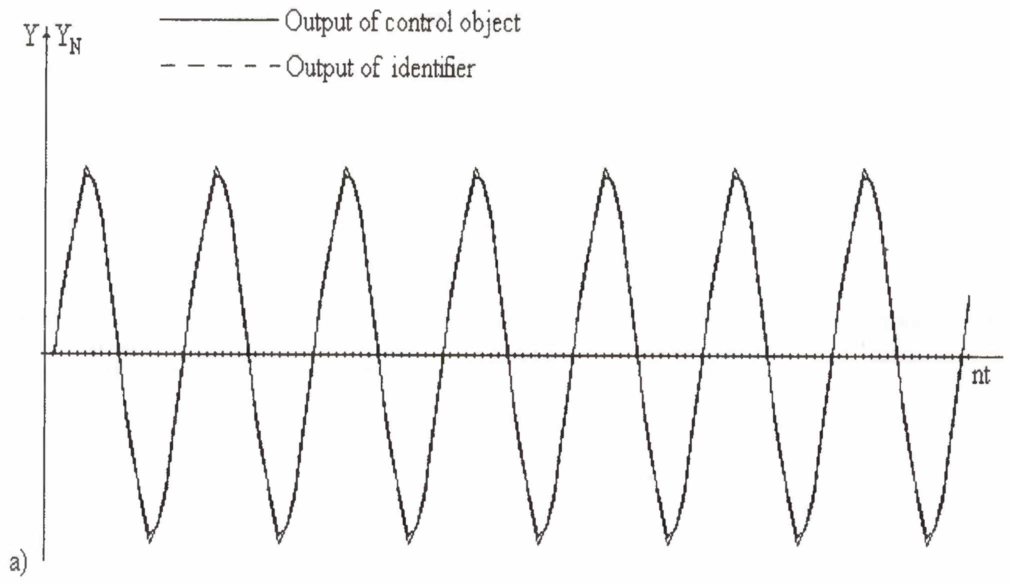


Figure 4.5.(a) : Simulation results of direct identification

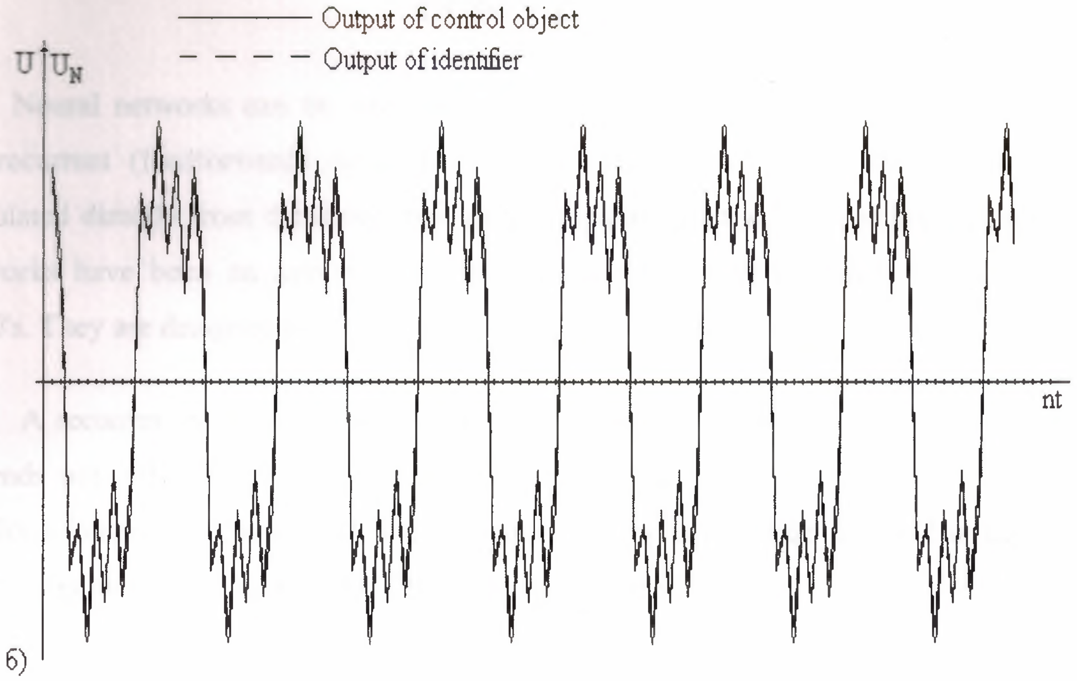
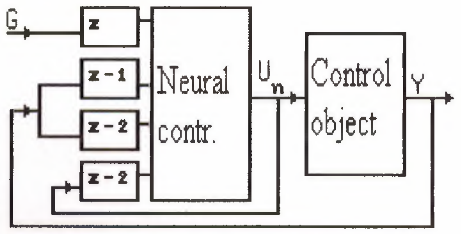


Figure 4.5.(b) : Simulation results of inverse identification

Also in fig. 4.6. 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.



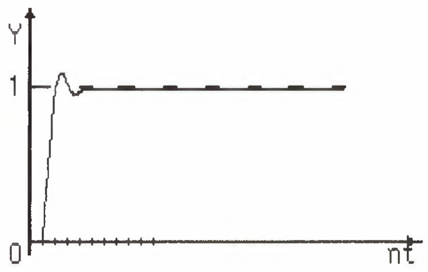


Figure 4.5. : Structure of inverse controller. **Figure 4.6. :** Time response of control system .

CONCLUSION

Neural networks can be classified into recurrent and nonrecurrent categories. Nonrecurrent (feedforward) networks have no feedback elements; the output is calculated directly from the input through feedforward connections. recurrent neural networks have been an important focus of research and development during the 1990's. They are designed to learn sequential or timevarying patterns.

A recurrent net is a neural network with feedback connections and the output depends not only on the current input to the network, but also on the current or previous outputs or states of the network .examples include Bam, Hopfield, Boltzmann machine,. and recurrent backpropagation nets.

And recurrent neural network techniques have been applied to a wide variety of problems. Simple partially recurrent neural networks were introduced in the late 1980's by several researchers including Rumelhart, Hinton, and Williams to learn strings of characters.

REFERENCES

[1] Rahib Abiyev. Recurrent Neural Network Systems for Control of Dynamic Plants. 2nd FAE International Symposium "Creating the Future". Lefke, TRNC, Turkey, 6-7 November, 2002.

[2] Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. of the National Academy of Sciences*, USA **97**, 2554-2558.

[3] Rahib Abiyev. Controller based on recurrent neural network for technological processes control. XXII International Congress on Operation Research and Industrial Engineering. OR/IE'01, Ankara Turkey, July 4-6, 2001,pp.65.

[4] Williams, R. and Zipser, D. (1996). Gradient-based learning algorithms for recurrent networks and their computational complexity, *Backpropagation: Theory, architectures and application*s, Edit. by Yves Chauvin and D. Rumelhart, Chap.13: 433-486.

[5] F.T.Aliew, R.R.Aliev, W.Steinmann, I.H.Muratov, R.H.Abiyev. Neural learning systems for technological processes control./ International Conference on Application of Fuzzy Systems and Softcomputing, ICAFS-98, Wiesbaden, Germany, October 5-7, 1998, pp.94-98.

[6] I.H.Muratov, R.H.Abiyev, R.R.Aliev. Learning systems based on neural networks for control technological processes //"Intelligent control and decision making systems".Thematic collected articles, N:1, Azerb.SOA,Baku-1998,pp.43-49(Russian)

[7] Hopfield, J.J., and Tank, D.W. (1985). "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, **52**, 141-152.

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

[9] Hertz, J, Krogh, A., and Palmer, R.G. (1991). *Introduction to the theory of neural computation.* Redwood City, Ca.: Addison-Wesley.

[10] Amit, D.J. (1989). *Modeling brain function. The world of attractor neural networks.* New York: Cambridge University Press.