NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

INTELLECTUAL SYSTEMS FOR TECHNOLOGICAL PROCESSES CONTROL

GRADUATION PROJECT
COM 400

STUDENT : SHAKIR TAJ

SUPERVISOR : Assist. Prof. Dr. RAHIB ABIVEV

LEFKOSA-2000

ACKNOWLEDGEMENTS

I express my sincere appreciation to all that helped in the way of preparation of this project.

First of all, I would like to thank my punctilious project supervisor **Assist. Prof. Dr. RAHIB ABIYEV**, whose attention to details and deadlines has compensated for my lack thereof. His encouraging words and kind admonitions have prodded me to action and was a key help in the making this project. Working without him is almost unthinkable.

I would also thank my erudite friends, whose help kept me on course, and whose words of encouragement kept me going.

The comments and suggestions by the following friends were also valuable: -

1. Mustanser Siddique
2. Atif Hussain Shah
3. Syed Kashif Hussain
4. Kamil Dimililer
5. Mian Sulaiman Saleem
6. Mudasser Siddique
7. Aneel Ahsan Khan.

I am very grateful to all my friends and my project supervisor Assist. Prof. Dr. Rahib Abivev for helping me in the most critical situation of my engineering academic carrier.

Above all, I thank the most gracious and the most merciful GOD for giving me the opportunity, ability, and stamina to prepare this project in time.

ABSTRACT

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

CONTENTS

Chapter 1
APPLICATIONS OF ARTIFICIAL SYSTEMS FOR SOLVING CONTROL PROBLEMS

The first chapter is about the applications, which are used by the control systems for solving control problems. Here are the discussions about :

1. Mobile Robot Control by a Structured Hierarchical Neural Network.
2. Identification and Control of Dynamical Systems Using Neural Networks.
3. Medical Ultrasound Imaging Using Neural Networks.
4. Electric Load Forecasting Using An Artificial Neural Network.
5. Operational Experience with A Neural Network In The Detection Of Explosives In Checked Airline Luggage.
6. Neural Network Models of Sensory Integration for Improved Vowel Recognition.

In all of these applications, the discussions will be about how we can make the use of Artificial Neural network in the making and working such type of devices. Thus these can be seen as follows:

1.1 Mobile Robot Control by a Structured Hierarchical Neural Network

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

1.2 Identification and Control of Dynamical Systems Using Neural Networks

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

1.3 Medical Ultrasound Imaging Using Neural Networks

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

1.4 Electric Load Forecasting Using an Artificial Neural Network

Artificial neural network (ANN) approaches to electric load forecasting. The ANN is used to learn the relationship among past, current and future temperatures and loads. In order to provide the forecasted load, the ANN interpolates among the load and temperature data in a training data set. The average absolute errors of the one-hour and 24-hour ahead forecasts in our test on actual utility data are shown to be 1.40% and 2.06%, respectively. This compares with an average error of 4.22% for 24-hour ahead forecasts with a currently used forecasting technique applied to the same data. Various techniques for power system load forecasting have been proposed in the last few decades. Load forecasting with lead-times, from a few minutes to several days, helps the system operator to efficiently schedule spinning serve allocation. In addition, load forecasting can provide information, which can be used, for possible energy interchange with other utilities. In addition to these economical reasons, load forecasting is also useful for system security. If applied to the system security assessment problem, it can provide valuable information to detect many vulnerable situations in advance[ ].

1.5 Operational Experience with A Neural Network In The Detection Of Explosives In Checked Airline Luggage

An Artificial Neural Network has been Implemented In the Explosives Detection Systems fielded at various airports. Tests of the on-line performance of the Neural Network (NN) confirmed Its superiority over standard statistical techniques, and the NN was installed as the decision algorithm In late October, 1989. Analysis of the mass of data being produced is still underway; but preliminary conclusions are presented.[ ].

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

1.6 Neural Network Models of Sensory Integration for Improved Vowel Recognition

Automatic speech recognizers currently perform poorly in the presence of noise. Humans, on the other hand, often compensate for noise degradation by extracting speech information from alternative sources and then integrating this information with the acoustical signal. Visual signals from the speaker's face are one source of supplemental speech information. We demonstrate that multiple sources of speech information can be integrated at a subsymbolic level to improve vowel recognition. Feedforward and recurrent neural networks are trained to estimate the acoustic characteristics; of the vocal tract from images of the speaker's mouth. These estimates are then combined with the noise-degraded acoustic information, effectively increasing the signal-to-noise ratio and improving the recognition of these noise-degraded signals. Alternative symbolic strategies, such as direct categorization of the visual signals into vowels, are also presented. The performances of these neural networks compared favorably with human performance and with other pattern-matching and estimation techniques[ ].

Communication by using the acoustic speech signal alone is possible, but often communication also involves visible gestures from the speaker's face and body. in situations where environmental noise is present or the listener is hearing impaired, these visual sources of information become crucial to understanding what has been said. Our ability to comprehend speech with relative ease under a wide range of environmental circumstances is due largely to our ability to fuse multiple sources of information in real time. Loss of information in the acoustic signal can be compensated for by using information about speech articulation from the movements around the mouth, or by Manuscript received October 16,1989; revised March 15,1990. This work was supported by using semantic information conveyed by facial expressions and other gestures. At the same time, the listener can use knowledge of linguistic constraints to further compensate for ambiguities remaining in the received speech signals.

Chapter 2
STRUCTURE AND LEARNING OF ARIFICIAL NEURAL NETWORK

An *artificial neural network* is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

2.1.Artificial Neural Networks Learning

A neural network is characterized by its pattern of connections between the neurons (called its *architecture),* its method of determining the weights on the connections (called its *training,* or *learning, algorithm),* and its *activation function.*

Since *what* distinguishes (artificial) neural networks from other approaches to information processing provides an introduction to both *how* and *when* to use neural networks, let us consider the defining characteristics of neural networks further.

A neural net consists of a large number of simple processing elements called *neurons, units, cells,* or *nodes.* Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve a problem. Neural nets can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

Each neuron has an internal state, called its *activation* or *activity level,* which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

For example, consider a neuron Y, illustrated in Figure 2. 1, that receives inputs from neurons X_1, X_2, and X_3. The activations (output signals) of these neurons are x_1, x_2 and x_3 respectively. The weights on the connections from X_1, X_2, and X_3 to neuron Y are w_1, w_2 and w_3, respectively. The net input, y_in, to neuron Y is the sum of the weighted signals from neurons X_1, X_2, and X_3. I.e.,

$$Y\_in = w_1 x_1 + w_2 x_2 + w_3 x_3 .$$

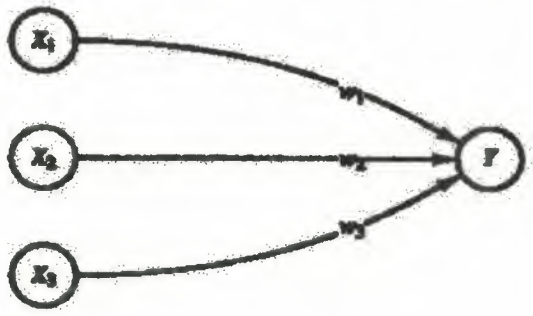


Figure 2.1 A simple (artificial) neuron.

The activation y of neuron Y is given by some function of its net input, $y = f(y\_in)$, e.g., the logistic sigmoid function (an S-shaped curve),

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

or any of a number of other activation functions. Now suppose further that neuron Y is connected to neurons Z_1 and Z_2, With weights v_1 and v_2 respectively, as shown in Figure1.2. Neuron Y sends its signal y to each of these units. However, in general, the values received by neurons Z_1 and Z_2 will be different, because each signal is scaled by the appropriate weight, v_1 or v_2. In a typical net, the activations of neurons Z_1 and Z_2 would depend on inputs from several or even many neurons, not just one, as shown in this simple example.

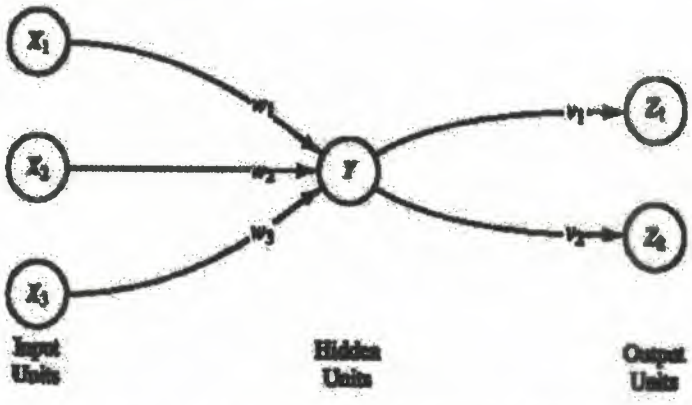


Fig. 2.2 A Very Simple Neural Network

Although the neural network in Figure 2.2 is very simple, the presence of a hidden unit, together with a nonlinear activation function, gives it the ability to solve many more problems than can be solved by a net with only input and output units. On-the other hand, it is more difficult to train (i.e., find optimal values for the weights) a net with hidden units. We also illustrate several typical activation functions and conclude the section with a summary of the notation we shall use throughout the rest of the text.

2.2 Neuron Models

Neural network models, even neuro-biological ones, assume many simplifications over actual biological neural networks. Such simplifications are necessary to understand the intended properties and to attempt any mathematical analysis. Even if all the properties of neurons were known, simplification would still be necessary for analytical tractability. A few models of neurons will be presented in this section.

2.2.1 McCulloc Pitts Model

McCulloch and Pitts modeled simple logical units called cells (or "neurons") so as to represent and analyze the logic of situations that arise in any discrete process in a computer, or anywhere else. Accordingly, the automata made up of these elementary units are usually called neural networks.

A McCulloch-Pitts cell is a very simple-two-state machine. From each cell emerges a single line or wire, called the output *fiber of the cell.* This output fiber may branch out after leaving the cell. Output fibers from different cells are not allowed to fuse together. Each branch must ultimately be fit as an input connection to another (or perhaps the same) cell. Two types of terminations are allowed. One provides an excitatory input, the other an inhibitory input. Any number of input connections to a cell is

permit to and gates may control the flow of information through a cell, which could be either of the excitatory or of the inhibitory type. The former is said to fire or activate the cell, whereas the latter is associated with the complementary function.

Each cell is a finite-state machine and accordingly operates in discrete time instants, which are assumed synchronous among all cells. At each moment, a cell is either firing or *quiet,* the two possible states of the cell. For each state there is an associated output signal, transmitted along the fiber branch of the cell. Since each cell has only two possible states, it is convenient to think of the firing state as producing a pulse whereas one may think of no *pulse* as the name of the signal associated with the quiet state. Cells change state as a consequence of the pulses received at their inputs. Each cell has a number associated with it, called the *threshold of* the cell. This threshold determines the state transition properties of-a cell C in the following manner. At any time index *k* there will be a certain distribution of activities in the fibers terminating upon C. we ignore all fibers that are quiet at this time and took to see if any inhibitory inputs are present. In the presence *of* one or more such inputs the cell C will not fire at time index (k + 1). This type of inhibition is referred to as *absolute* inhibition because a cell will fire at time index (k + 1) *if* and only if, at time index k, the number of active excitatory inputs equals or exceeds the threshold and *no inhibitor is active.*

The McCulloch-Pitts network (called *net* for the sake of brevity) model is based on several simplifying assumptions. First, the state of a cell at time index *(k + 1)* is assumed to depend on its state at time index *k* and on any input at that same instant of time. Second, the inhibition is taken to be absolute in the sense that a single inhibitory signal can block response of a cell to any amount of excitation. One in which a cell fires if the difference between the amounts of excitation and inhibition exceeds the threshold can replace this assumption. In fact the reader is advised to solve the relevant problems at the end of this chapter using this last assumption. Finally, a standard delay between the input and output of each cell is assumed. Since the individual cells or neurons operate on the same time scale, the overall network operation is synchronous. Biological neurons, however, operate on different time scales and therefore function asynchronously.

In this type of network, any kind of more or less permanent memory must depend on the existence of closed or feedback paths. Otherwise, in the absence of external stimulation, all activity must soon die out, leaving all cells in the quiet state. *A feedback fiber* runs from the output fiber *of* a cell back to an excitatory termination at the input *of* the very same cell. Once this cell has been fired (by a signal from the *start* fiber), it will continue to fire at all successive-time instants until it is halted by a signal on the inhibitory *stop* fiber. Throughout this interval of activity, it will send pulses to the *gate* cell and permit the passage of information.

Each neural network built from McCulloch-Pitts cells is a finite-state machine, and every finite-state machine is equivalent to and can be simulated by some neural network. At any moment, the global state of the net is given by the firing pattern of its cells. Let *x(k),* u(k), and *y(k)* denote, respectively, the state; input, and output vectors at time index k. The state vector is composed of the state of each neuron at a certain time index as given by its last output. For suitable functions, F(.) and *G(.),* the state-transition equation

$$\mathbf{x}(k+1) = \mathbf{F}(\mathbf{x}(k), \mathbf{u}(k))$$

Is determine d by the connection structure of the network, and the output vector

$$y(k+1) = \mathbf{G}\,(\,\mathbf{x}(k), u(k))$$

is determined by the fibers that carry output signals.

A further development of the work of McCulloch and Pitts led to threshold logic units (TLUs) with adjustable weights. A TLU is a device **with n inputs,** $x_1, x_2, x_3, \ldots x_n$, and an output y. There are n

+ 1 parameters, namely the weights ($w_1, w_2, w_{3,...,} w_{n,}$) and a threshold θ. The TLU computes an output value at discrete time indices $k = 1, 2$, according to

$$y(k+1) = \begin{cases} 1....if..\sum_{i=1}^{n} w_i x_i(k) \geq \theta \\ 0.....otherwise. \end{cases} \qquad (2.1)$$

Positive weights $w_i > 0$ represent excitatory synapses, whereas negative weights $w_i < 0$ represent inhibitory ones. Note the unit time index delay occurring between the instants when the inputs are applied and the output appears. A TLU is sketched in Fig.2.3. A McCulloch-Pitts neuron is governed by the threshold decision rule of a TLU, shown in Eq. (2.1), or of its bipolar variant (in -which case 1 replaces the output state 0). Equation (2. 1) and its variants are popular because of their mathematical tractability. They fan, however, to capture the stochastic spatial and temporal complexities inherent in neuronal information processing. The reader is referred to ref. [22] for further details on this matter. In applying this model the output of a TLU is often assumed to belong to the set $\{\alpha, 1\}$, which represents the binary states of a neuron, where α is a nonzero number. The value of a may be either - 1, in the bipolar case, or a small positive number, e.g., 0. 1, in the modified unipolar case. This choice will lead to faster convergence in almost all learning algorithms, because if a = 0 (the strict unipolar case), the updates of a weight connected to the ouput of a neuron will become zero, whereas making α nonzero ensures that a weight will be updated whenever it will be required.

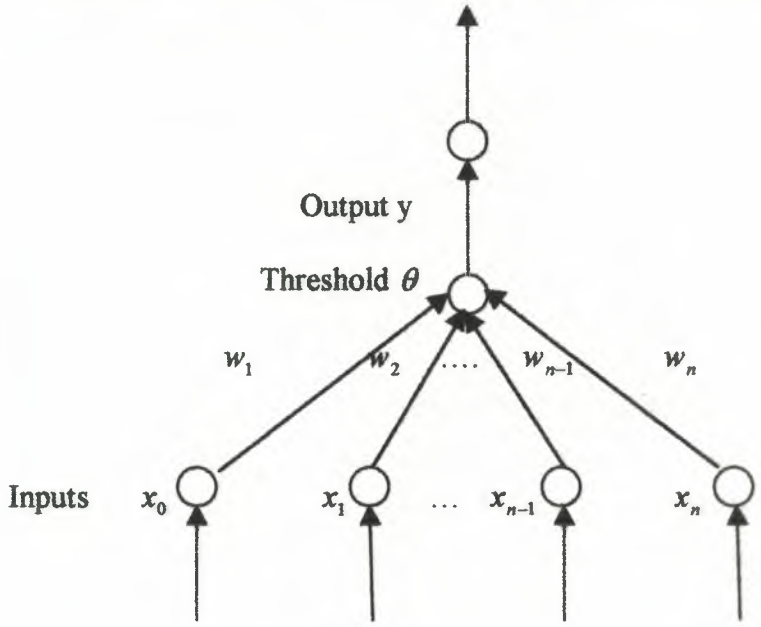


FIGURE 2.3 A threshold logic unit (ILU), whose transfer characteristics described in Eq. (2.1). Processing takes place in the unit whose threshold is shown. The other units are shown only for complete and are really not needed in the model of a TLU.

This model of a neuron is highly simplified. A network of these very simple neurons can compute any logical (Boolean) function Indeed, with properly chosen weights. A network of TLUs can simulate any digital computer. That is, a network of simple neurons can do at least what a modem digital computer can.

2.2.2 Neuron Models with Continuous Transfer Characteristics

The action potential or spike of a biological neuron is a continuous variable. The time instant at which a postsynaptic potential change occurs affects its interaction with other potential changes. Information is not encoded only by the excitation and resting states of neurons; the rate at which a neuron is excited also carries information. Real neurons have integrative time delays due to capacitance. The time evolution of a real neuron is better described by differential equations instead of the discrete time transition equations for TLUs. Let us examine a widely used neuron model that includes some of the foregoing considerations.

Instantaneous input x1 to the fifth neuron is defined to approximate the mean some potential from-the effects of its excitatory and inhibitory synapses as well as its threshold. If there are no external inputs and leakage current is ignored, then with the outputs y_j for $j = 1,2,..n$ from n neurons used as inputs to the *ith* neuron (having a threshold θ_i) after multiplication by connection weights wipe the value of x_i is

$$x_i = \sum_{j=1}^{n} w_{ij} y_j - \theta_i \qquad (2.2)$$

The output y_j of a neuron represents the short-term average of the firing rate of neuron j and is given by

$$y_j = f(\lambda x_j) \qquad (2.3)$$

Where λ is a positive number. The unipolar or logsigmoid form of the input/output (transfer) characteristic f (-) is described by

$$y_i = \frac{1}{1+\exp(-\lambda x_i)} = \frac{1}{1+\exp[-\lambda(\sum_{j=1}^{n} w_{ij} y_j - \theta_i} \qquad (2.4)$$

And it approaches the characteristic of a unipolar TLU as A tends to ∞. The transfer characteristic may also be defined (for the bipolar case) through

$$y_i = \frac{2}{1=\exp(-\lambda x_i)} - 1 = \frac{2}{1+\exp[-\lambda(\sum_{j=1}^{n} w_{ij} y_j - \theta_i)]} - 1 \qquad (2.5)$$

In Eq. (2.5), the limiting values of y_j as λ approaches - are either +1 or -1, depending upon whether x_i is positive or negative. This characteristic is referred to as tansigmoid.

A more realistic model includes the time delay due to membrane capacitance C_i and leakage current through the transmembrane resistance R_i. The dynamics of such a neuron model are described by the following equations:

$$C_i \frac{\partial x_i}{\partial t} = \sum_j w_{ij} - \frac{x_i}{R_i} + I_i \qquad (2.6)$$

$$x_i = f_i^{-1}(y_i),$$

Where 1i is the external current stimulus to neuron i. Note that the weight w_{ij} has the dimension of transconductance, which is realizable by an active device. Therefore, active devices model synapses in this case.

Figure 2. 4 illustrate the parameters of this model superimposed on a neuron. An electrical circuit simulating Eq. (2.6) is shown in Fig. 2. 5. The weights w_{ij} are realized as transconductances. Each operational amplifier has two outputs, y_i and $-y_i$ to avoid negative weights for inhibitory links. An inhibitory input to neuron *i* from neuron *j* is linked by a positive weight to the -*y* output of neuron *j*.

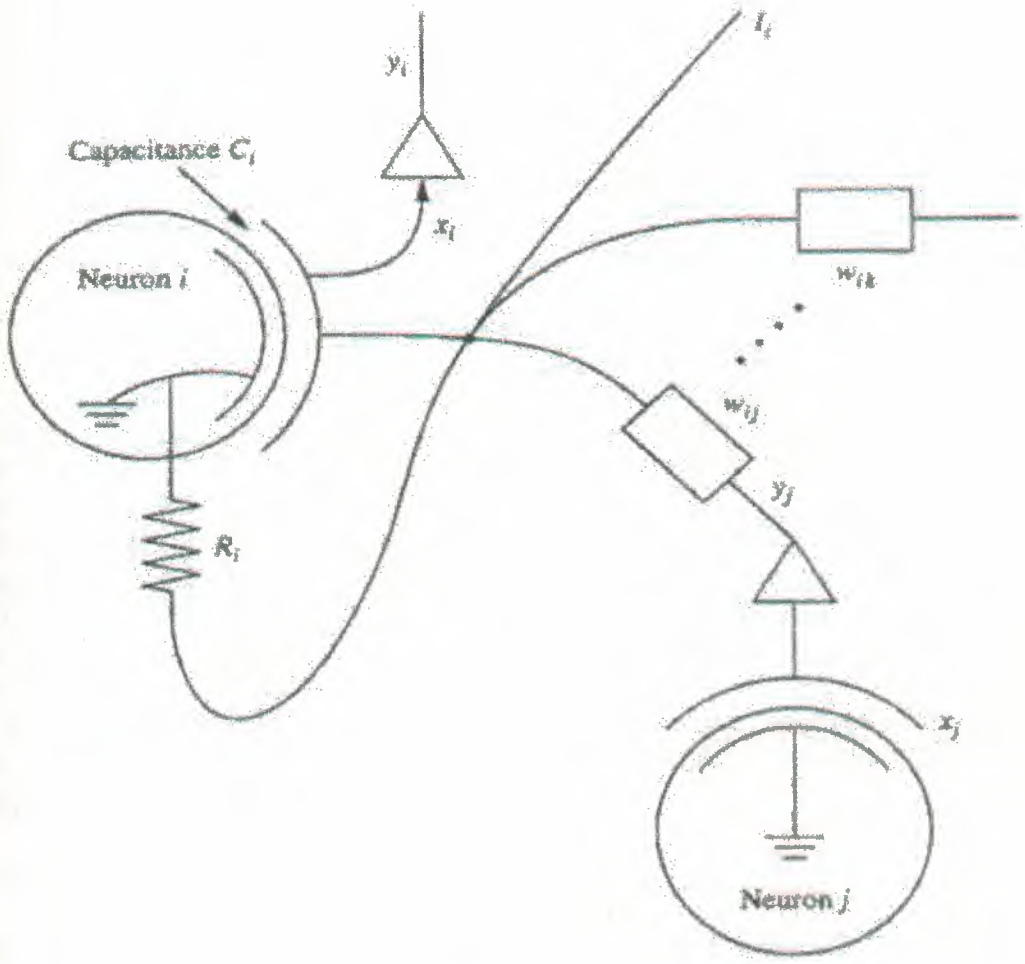


Fig. 2.4 Model showing how the dynamics of the *i*th neuron are coupled with the output y_j from the *j*th nouron through the connection weight w_{ij}, external current I_i, etc.

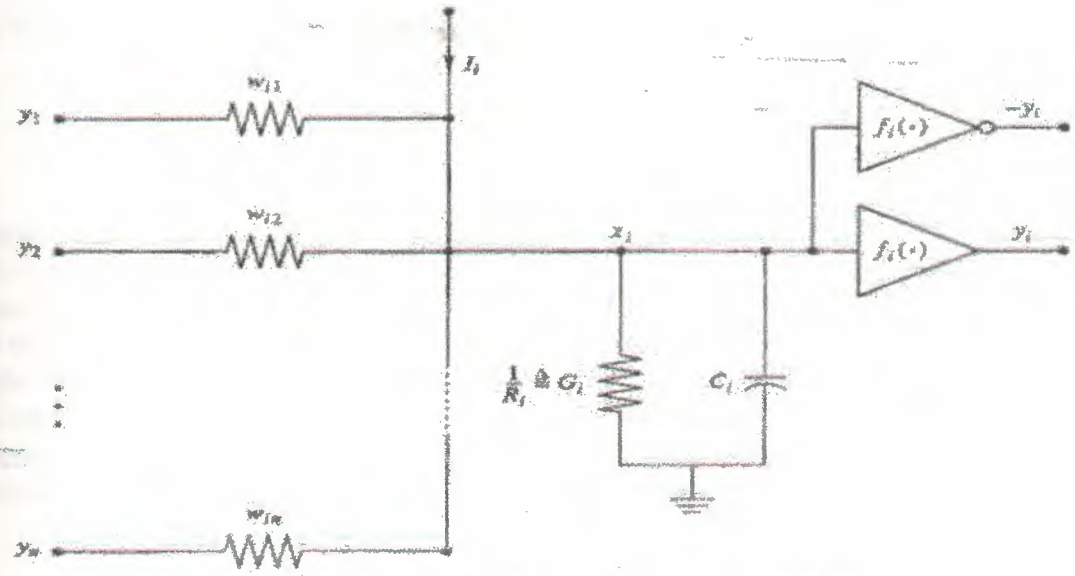


Fig 2.5 The Electric Circuit simulating Eq. (2.6)

2.2.3 Common Activation Functions

A neural network is characterized by its pattern of connections between the neurons (called its *architecture),* its method of determining the weights on the connections (called its *training,* or *learning, algorithm),* and its *activation function.*

The basic operation of an artificial neuron involves summing its weighted input signal and applying an output, or activation, function. For the input units, this function is the identity function (see Figure2.6). Typically, the same activation function is used for all neurons in any particular layer of a neural net, although this is not required. In most cases, a nonlinear activation function is used. In order to achieve the advantages of multilayer nets, compared with the limited capabilities of single-layer nets, nonlinear functions are required (since the results of feeding a signal through two or more layers of linear processing elements-i.e., elements with linear activation functions-Are no different from what can be obtained using a single layer).

(i) Identity function:

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

Single-layer nets often use a step function to convert the net input, which is a continuously valued variable, to an output unit that is a binary (1 or 0) or bipolar (1 or - 1) signal (Fig 2.7). The binary step function is also known as the threshold function or Heaviside function.

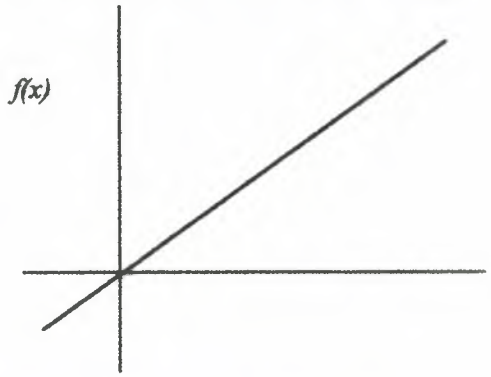


Figure 2.6 Identity function.

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

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

Sigmoid functions (S-shaped curves) are useful activation functions. The logistic function and the hyperbolic tangent functions are the most common. They are especially advantageous for use in neural nets trained by backpropagation, because the simple relationship between the value of the function at a point and the value of the derivative at that point reduces the computational burden during training.
The logistic function, a sigmoid function with range from 0 to 1, is often used as the activation function for neural nets in which the desired output values either are binary or are in the interval between 0 and 1. To emphasize the range of the function, we will call it the *binary sigmoid;* it is also called the *logistic sigmoid.* This function is illustrated in Figure 2.8 for two values of the *steepness parameter* σ .

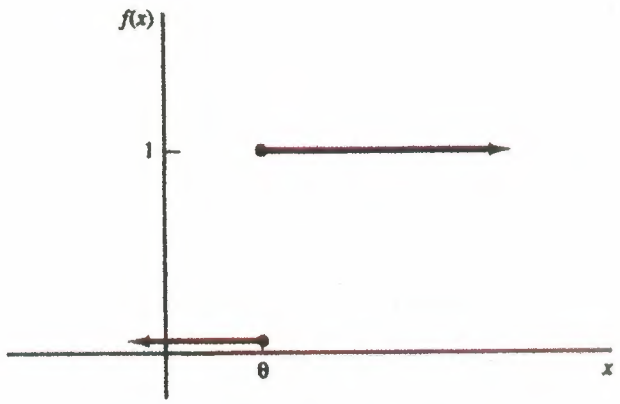


Fig 2.7 Binary Step Function

(iii) Binary sigmoid:

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

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

As the logistic sigmoid function can be scaled to have any range of values that is appropriate for a given problem. The most common range is from - 1 to 1, we call this sigmoid the *bipolar sigmoid.* It is illustrated in Figure 2.8 for $\sigma = 1$.

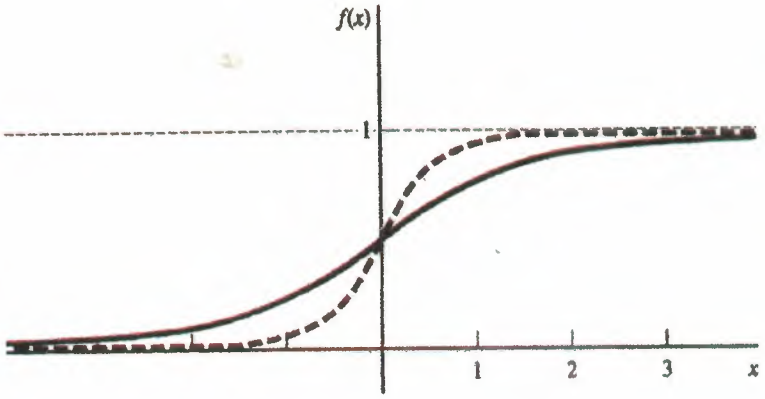


Fig. 2.8 Binary Sigmoid.
Steepness parameters $\sigma = 1$ and $\sigma = 3$.

iv) Bipolar sigmoid:

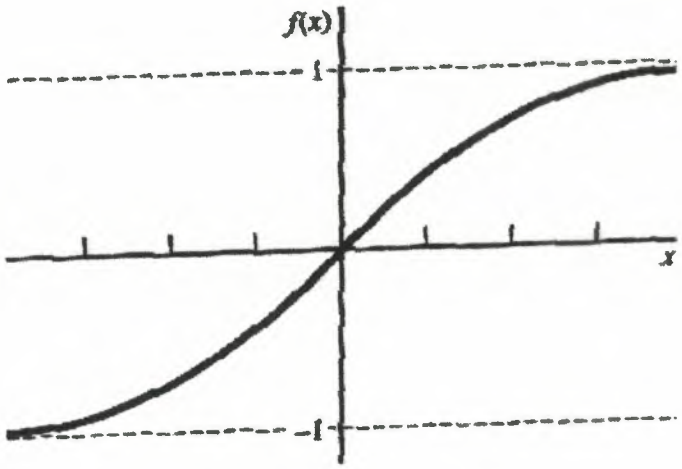


Fig.2.9 Bipolar Sigmoid.

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

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

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

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

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

The hyperbolic tangent is

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

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

The derivative of the hyperbolic tangent is

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

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

2.3 Typical Architectures of Recurrent Neural Network

Several neural networks have been developed to learn sequential or time-varying patterns. Unlike the recurrent nets with symmetric weights or the feedforward nets, these nets do not necessarily produce a steady-state output. In this section, we consider a simple recurrent net [Elman, 1990; Hertz, Krogh, & Palmer, 1991] that can be used to learn strings of characters [Servan-Schreiber, Cleeremans, & McClelland, 1989]. This net can be considered a "partially recurrent" net, in that most of the connections are feedforward only. A specific group of units receives feedback signals from the previous time step. These units are known as *context* units. The weights on the feedback connections to the context units are fixed, and information processing is sequential in time, so training is essentially no more difficult than for a standard backpropagation net. The architecture for a simple recurrent net is as shown in Figure 2.10.

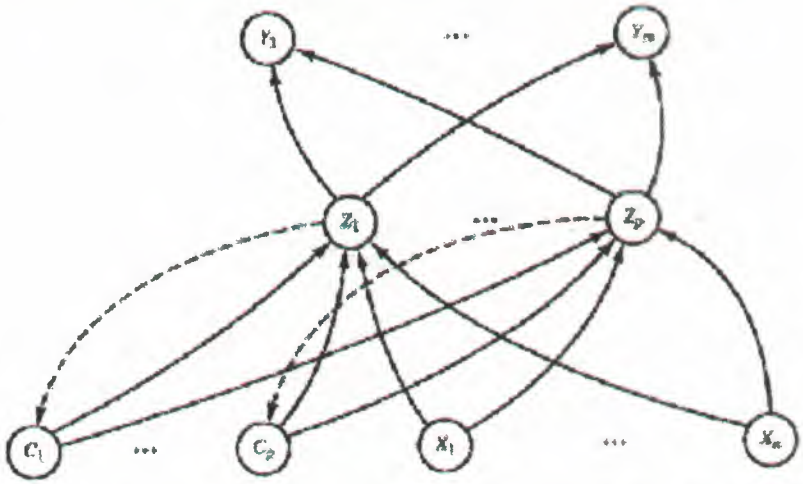


Fig 2.10 Architecture for Simple Recurrent Neural Net.

Consider some of the fundamental features of how neural networks operate. Detailed discussions of these ideas for a number of specific nets are presented. The building blocks of our examination here are the network architectures and the methods of setting the weights (training). This also illustrate several typical illustration functions and conclude the section with a summary of notation that are used throughout the rest of the text.

Often, it is convenient to visualize neurons as arranged in layers. Typically, neurons in the same layer behave in the same manner. Key factors in determining the behavior of a neuron ' are its activation function and the pattern of weighted connections over which it sends and receives signals. Within each layer, neurons usually have the same activation function and the same pattern of connections to other neurons. To be more specific, in many neural networks, the neurons within a layer are either fully interconnected or not interconnected at all. If any neuron in a layer (for instance, the layer of hidden units) is connected to a neuron in another layer (say, the output layer), then each hidden unit is connected to every output neuron.

The arrangement of neurons into layers and the connection patterns within and between layers is called the *net architecture*. Many neural nets have an input layer in which the activation of each unit is equal to an external input signal. Neural nets are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnect links between the slabs of neurons. This view is motivated by the fact that the weights in a net contain extremely important information. The net shown in Figure 2.2 has two layers of weights.

The single-layer and multilayer nets illustrated in Figures 2.11 and 2.12 are examples of *feedforward* net-nets in which the signals flow from the input units to the output units, in a forward direction. The fully interconnected competitive net in Figure 2.13 is an example *of* a *recurrent* net, in which there are closed-loop signal paths from a unit back to itself.

2.3.1 Single-Layer Net

A single-layer net has one layer of connection weights. Often, the units can be distinguished as input units, which receive signals from the outside world, and output units, from which the response of the net can be read. In the typical singlelayer net shown in Figure 2.11, the input units are fully connected to output units but are not connected to other input units, and the output units are not connected to other output units. By contrast, the Hopfield net architecture, shown in Figure 2.11, is an example of a single-layer net in which all units function as both input and output units.

For pattern classification, each output unit corresponds to a particular category to which an input vector may or may not belong.

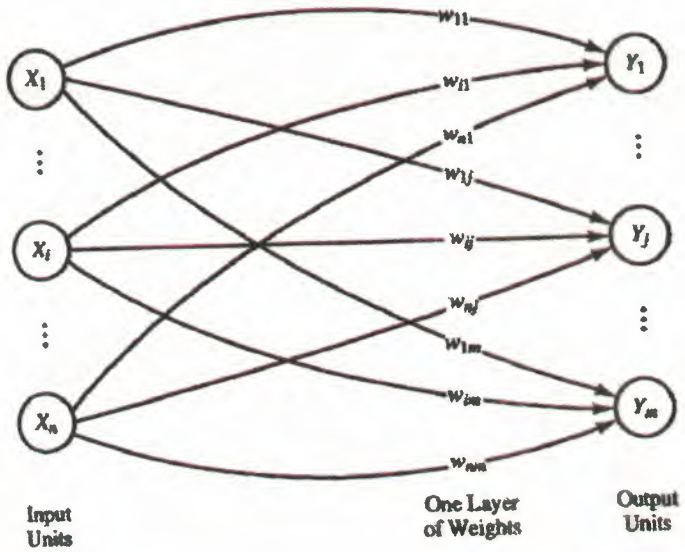


Figure 2.11 A single layered Network

For a single layer net, the weights for one output unit do not influence the weights for other output units. For pattern association, the same architecture can be used, but now the overall pattern of output signals gives the response pattern associated with the input signal, that caused it to be produced. These two examples illustrate the fact that the same type of net can be used for different problems, depending on the interpretation of the response of the net.

2.3.2 Multilayer net

A multilayer net is a net with one or more layers (or levels) of nodes (the so called hidden units) between the input units and the output units. Typically, there is a layer of weights between two adjacent levels of units, (input, hidden, or output).

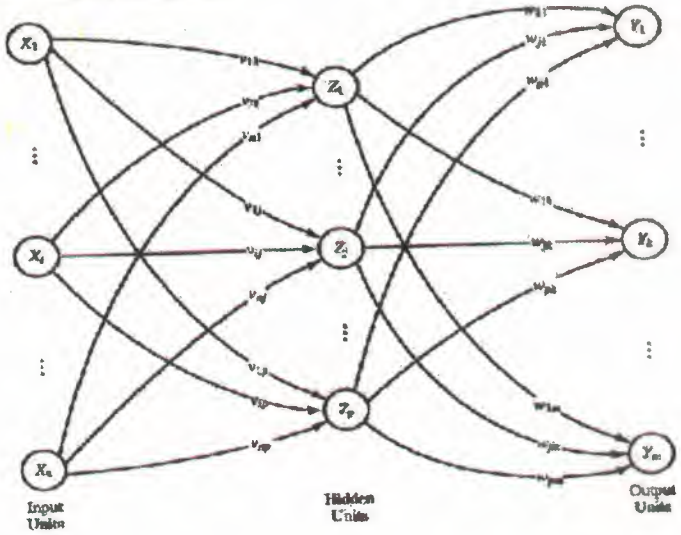


Fig 2.12 A Multilayer Network

Multilayer nets can solve more complicated problems than, can single-layer nets, but training may be more difficult. However, in some cases, training may be more successful, because it is possible to solve a* problem that a single-layer net cannot, be trained to perform correctly at all.

2.3.3 Competitive layered Net

A competitive layer forms a part of a large number of neural networks. Typically, the inter-connections between neurons in the competitive layer are not shown in the architecture diagrams for

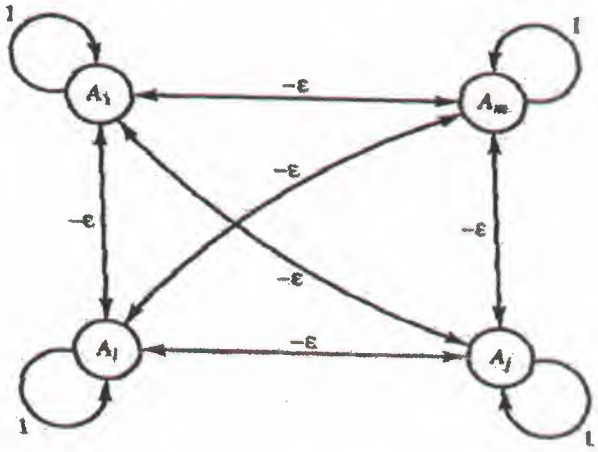


FIG 2.13 A Competitive Network.

such nets. An example of the architecture for a competetive layer is given in Figure 2.13; the competitive interconnections have weights of - ε .

2.4 Learning of The Neural Networks

In addition to the architecture, the method of setting the values of the weights (training) is an important distinguishing characteristic of different neural nets. For convenience, we shall distinguish two types of training-supervised and unsupervised-for a neural network; in addition, there are nets whose weights are fixed without an iterative training process.

Many of the tasks that neural nets can be trained to perform fall into the areas of mapping, clustering, and constrained optimization. Pattern classification and pattern association may be considered special forms of the more general problem of mapping input vectors or patterns to the specified output vectors or patterns.

There is some ambiguity in the labeling of training method s as supervised or unsupervised, and some authors find a third category, self-supervised training, useful. However, in general, there is a useful correspondence between the type of training that is appropriate and the type of problem we wish to solve. We summarize here the basic characteristics of supervised and unsupervised training and the types of problems for which each, as well as the fixed-weight nets, is typically used.

2.4.1 Supervised training

In perhaps the most typical neural net setting, training is accomplished by presenting a sequence of training vectors, or patterns., each with an associated target output vector. The weights are then adjusted according to a learning algorithm. This process is known as *supervised training.*

Pattern association is another special form of a mapping problem, one in which the desired output is not just a "yes" or "no," but rather a pattern. A neural net that is trained to associate a set of input vectors with a corresponding set of output vectors is called an associative memory. If the desired output vector is the same as the input vector, the net is an autoassociative memory; if the output target vector is different from the input vector, the net is a *heteroassociative* memory. After training, an associative memory can recall a stored pattern when it is given an input vector that is sufficiently similar to a vector it has learned.

Multilayer neural nets can be trained to perform a nonlinear mapping from an n-dimensional space of input vectors (n-tuples) to an m-dimensional output space-i.e., the output vectors are m-tuples.

2.4.2 Unsupervised training

Self-organizing neural nets group similar input vectors together without the use of training data to specify what a typical member of each group looks like or to which group each vector belongs. A sequence of input vectors is provided, but no target vectors are specified. The net modifies the weights so that the most similar input vectors are assigned to the same output (or cluster) unit. The neural net will produce an exemplar (representative) vector for each cluster formed.

Unsupervised learning is also used for other tasks, in addition to clustering.

2.5 Back Propagation in time.

Back Propagation is now the most widely used tool in the field of Artificial neural networks. At the core of backpropagation is a method for calculating derivatives exactly and efficiently in any large system made up of elementary subsystems or calculations which are represented by known, differentiable functions; thus, backpropagation has many applications which do not involve neural networks as such.

This paper first reviews basic backpropagation, a simple method which is now being widely used in areas like pattern recognition and fault diagnosis. Next, it presents the basic equations for back-propagation through time, and discusses applications to areas like pattern recognition involving dynamic systems, systems identification, and control. Finally, it describes further extensions of this method, to deal with systems other than neural networks, systems involving simultaneous equations or true recurrent networks, and other practical issues which arise with this method Pseudocode is provided to clarify the algorithms. The chain rule for ordered derivatives,the theorem which underlies backpropagation is briefly discussed.

2.5.1 Structure of Back Propagation

Backpropagation through time is a very powerful tool, with applications to pattern recognition, dynamic modeling, sensitivity analysis, and the control of systems over time, among others. It can be applied to neural networks, to econometric models, to fuzzy logic structures, to fluid dynamics models, and to almost any system built up from elementary subsystems or calculations. The one serious constraint is that the elementary subsystems must be represented by functions known to the user, functions which are both continuous and differentiable (i.e., possess derivatives). For example, the first practical application of backpropagation was for estimating a dynamic model to predict nationalism and social communications in 1974 [1].

At its core, backpropagation is simply an efficient and exact method for calculating all the derivatives of a single target quantity (such as pattern classification error) with respect to a large set of input quantities (such as the parameters or weights in a classification rule). Backpropagation through time extends this method so that it applies to dynamic systems. This allows one to calculate the derivatives needed when optimizing an iterative analysis procedure, a neural network with memory, or a control system which maximizes performance over time.

A simple example of a backpropagation in time net is shown in Figure 2.14. An expanded form of a backpropagation in time net is illustrated in Figure 2.15. A generalization of this allows for both external inputs and recurrent signals from the previous time step, as shown in Figure 2.16. As in the simple recurrent net discussed in the previous section, the recurrent connections have weights fixed at 1; the adjustable weights are shown.

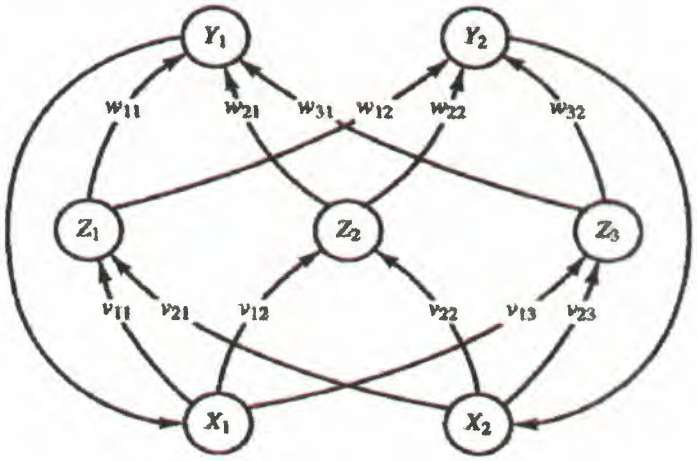


Figure 2.14 A recurrent multilayer net in which the outputs at one time step become inputs at the next step.

The training algorithm using backpropagation in time for a recurrent net of the form illustrated in Figure 2.14 or 2.16 is based on the observation that the performance of such a net for a fixed number of time steps N is identical to the results obtained from a feedforward net with $2N$ layers of adjustable

weights. For example, the results produced by the net in Figure 2.16 after three time steps could also be obtained from the net shown in Figure 2.15.

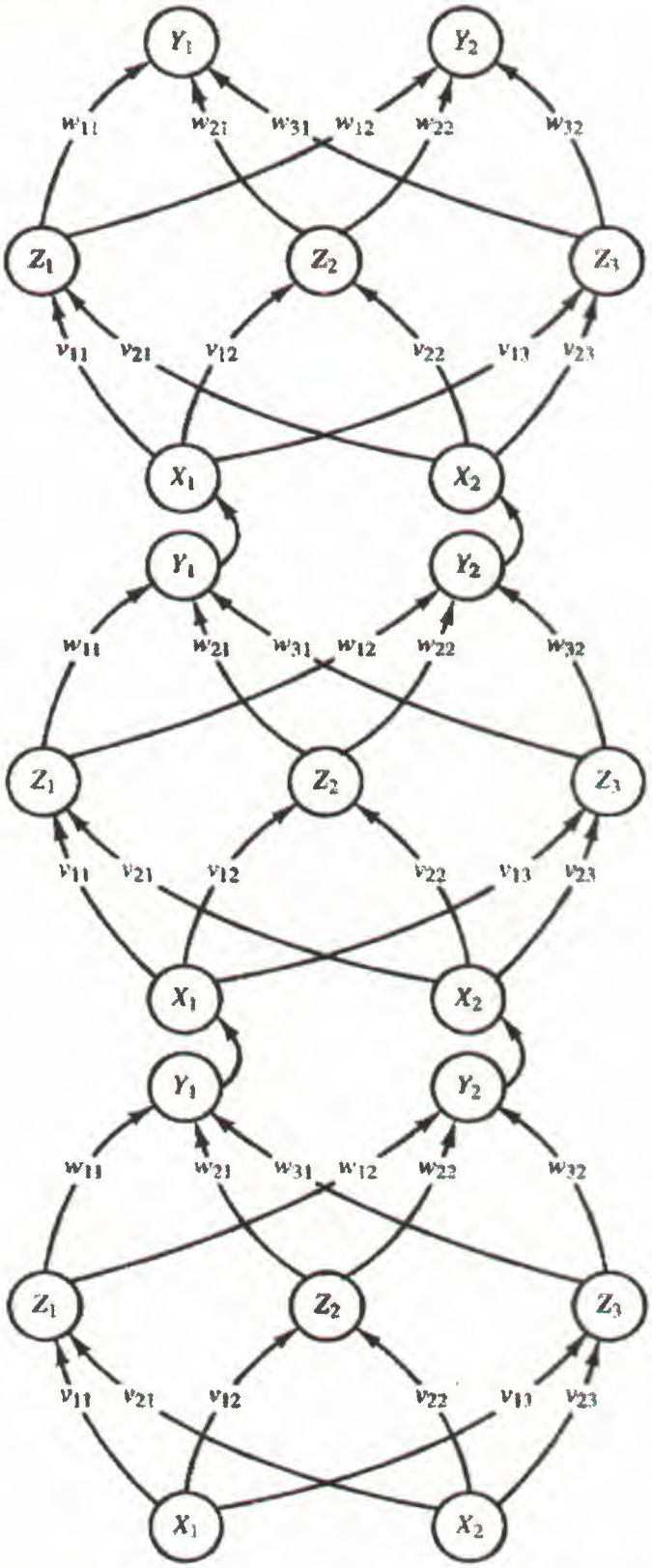


Fig. 2.15 The recurrent multilayer net of Figure 2.16 explained for three time steps.

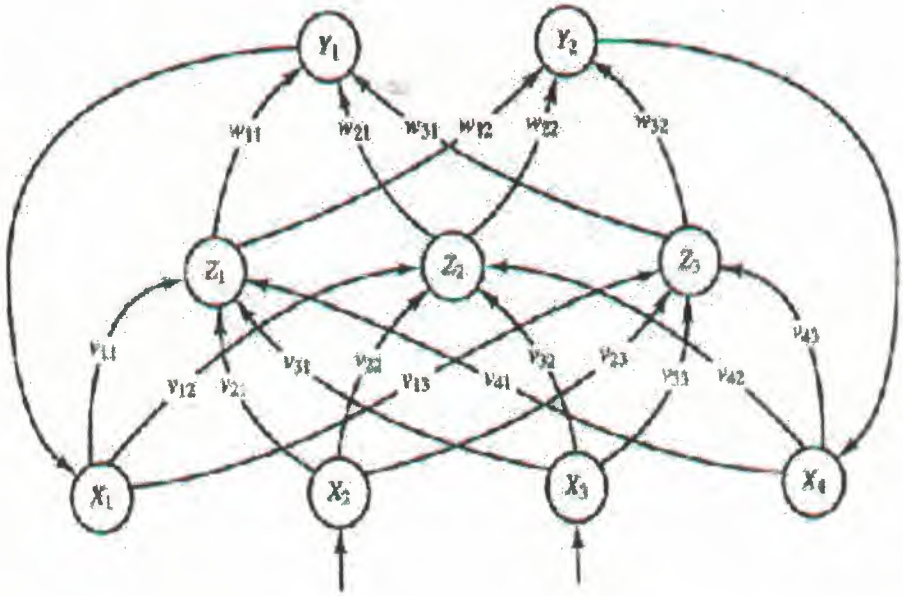


Figure 2.16 A recurrent multilayer net with external and recurrent inputs at each step.

The training process consists of a feedforward pass through the entire expanded network (for the desired number of time steps). The error is then computed for each output layer (i.e., for each time step). The weight adjustments for each copy of the net are determined individually (i.e., computed) and totaled (or averaged) over the number of time steps used in the training. Finally, all copies of each weight are updated. Training continues in this way for each training pattern, to complete an epoch. As with standard backpropagation, typically, many epochs are required.

Note that it is not necessary actually to simulate the expanded form of the net for training. The net can run for several time steps, determining the information on errors and the weight updates at each step and then totaling the weight corrections and applying them after the specified number of steps.

In addition, information on errors does not need to be available for all output units at all time steps. Weight corrections are computed using whatever information is available and then are averaged over the appropriate number of time steps. In the example in the next section, information on errors is supplied only at the second time step; no responses are specified after the first time step.

2.5.2 The Supervised Learning Problem

Basic backpropagation is current the most popular method for performing the supervised learning task, which is symbolized in Fig 2.17.

In supervised learning, we try to adapt an artificial neural network so that its actual outputs ($\hat{Y}$) come close to some target outputs (Y) for a training set which contains T patterns. The goal is to adapt the parameters of the network so that it performs well for patterns from outside the training set. The main use of supervised learning today lies in pattern recognition work. For example, suppose that we are trying to build a neural network which can learn to recognize handwritten ZIP codes. (AT&T has actually done this [13], although the details are beyond the scope of this paper). We assume that we already have a camera and preprocessor which can digitize the image, locate the five digits, and provide a 19 x 20 grid of ones and zeros representing the image of each digit. We want the neural network to input the 19 x 20 image, and output a classification; for example, we might ask the network to output four binary digits which, taken together, identify which decimal digit is being observed.

Before adapting the parameters of the neural network, one must first obtain a training database of actual handwritten digits and correct classifications. Suppose, for example, that this database contains 2000 examples of handwritten digits. In that case, T = 2000. We may give each example a label *t* between 1 and 2000. For each sample *t*, we have a record of the input pattern and the correct classification. Each input pattern consists of 380 numbers, which may be viewed as a vector with 380 components; we may call this vector ***X(t)***. The desired classification consists of four numbers, which may be treated as a vector ***Y(t)***. The actual output of the network will be $\hat{Y}(t)$, which may differ from the desired output ***Y(t)***, especially in the period before the network has been adapted. To solve the supervised learning problem, there are two steps:

- We must specify the "topology" (connections and equations) for a network which inputs ***X(t)*** and outputs a four-component vector $\hat{Y}(t)$, an approximation to ***Y(t)***. The relation between the inputs and outputs must depend on a set of weights (parameters) IN which can be adjusted.
- We must specify a "learning rule", a procedure for adjusting the weights ***W*** so as to make the actual outputs $\hat{Y}(t)$ approximate the desired outputs ***Y(t)***.

Basic backpropagation is currently the most popular learning rule used in supervised learning. It is generally used with a very simple network design-to be described in the next section-but the same approach can be used with any network of differentiable functions.

Even when we use a simple network design, the vectors ***X(t)*** and ***Y(t)*** need not be made of ones and zeros. They can be made up of any values which the network is capable of inputting and outputting. Let us denote the components of ***X(t)*** as $X_1(t).....X_m(t)$ so that there are ***m*** inputs to the network. Let us denote the components of ***Y(t)*** as $Y_{1(t)}.....Y_n(t)$ so that we have ***n*** outputs. Throughout this paper, the components of a vector will be represented by the same letter as the vector itself, in the same case; this convention turns out to be convenient because ***x(t)*** will represent a different vector, very closely related to ***X(t)***.

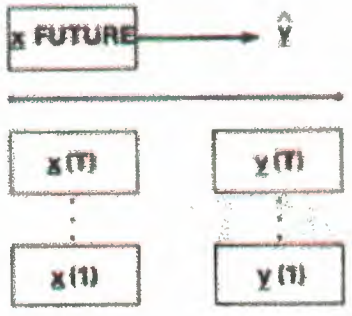


Fig. 2.17 illustrates the supervised learning task in the general case. Given a history of *X(1)* ... *X (T)* and *Y(1)* ... *Y(T)*, we want to find a mapping from X to Y which will perform well when we encounter new vectors *X* outside the training set. The index *"t"* may be interpreted either as a time index or as a pattern number index; however, this will not assume that the order of patterns is meaningful.

Simple Feedforward Networks

Before we specify a learning rule, we have to define exactly how the outputs of a neural net depends on its inputs and weights. In basic backpropagation, we assume the following logic:

$$x_i = X_i, \qquad 1 \le i \le m \qquad (2.7)$$

$$net_i = \sum_{j=1}^{i-1} W_{ij} x_i, \qquad m < i \le N+n \qquad (2.8)$$

$$x_i = s(net_i), \qquad m < i \le N+n \qquad (2.9)$$

$$Y_i = x_{i+N}, \qquad 1 \le i \le n \qquad (2.10)$$

where the functions in *(2.9)* is usually the following sigmoidal function:

$$s(z) = \frac{1}{1+e^{-z}}$$

and where N is a constant which can be any integer you choose as long as it is no less than m. The value of $N + n$ decides how many neurons are in the network (if we include inputs as neurons). Intuitively, net_i represents the total level of voltage exciting a neuron, and x_i represents the intensity of the resulting output from the neuron. (x_i is sometimes called the "activation level" of the neuron.) It is conventional to assume that there is a threshold or constant weight W_{io} added to the right side of *(2.8)*, however, we can achieve the same effect by assuming that one of the inputs (such as X_m) is always 1.

The significance of these equations is illustrated in Fig.2.18. There are $N + n$ circles, representing all of the neurons in the network, including the input neurons. The first m circles are really just copies of the inputs $X_1 \ldots X_m$; they are included as part of the vector x only as a way of simplifying the notation.

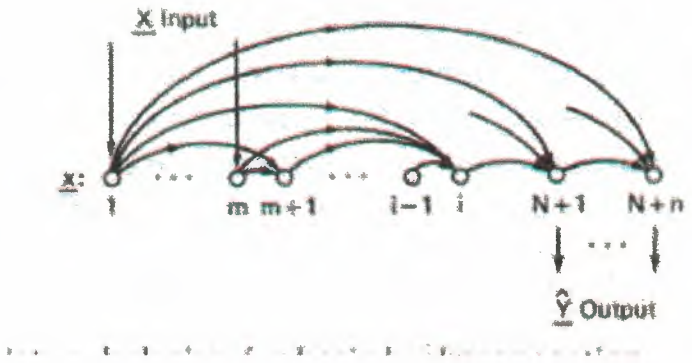


Fig. 2.18 Network design for basic backpropagation

Every other neuron in the network such as neuron number i, which calculates net_i and x_i takes input from every cell which precedes it in the network. Even the last output cell, which generates $\hat{Y}_n$, takes input from other output cells, such as the one which outputs $\hat{Y}_{n-1}$.

In neural network terminology, this network is "fully connected" in the extreme. As a practical matter, it is usually desirable to limit the connections between neurons. This can be done by simply fixing some of the weights W_{ij} to zero so that they drop out of all calculations. For example, most researchers prefer to use "layered" networks, in which all connection weights W_{ij} are zeroed out, except for those going from one "layer" (subset of neurons) to the next layer. In general, one may zero

out as many or as few of the weights as one likes, based on one's understanding of individual applications. For those who first begin this work, it is conventional to define only three layers-an input layer, a "hidden" layer, and an output layer. This section will assume the full range of allowed connections, simply for the sake of generality.

2.5.3 Calculating Derivatives: Theoretical Background

Many papers on backpropagation suggest that we need only use the conventional chain rule for partial derivatives to calculate the derivatives of E with respect to all of the weights. Under certain conditions, this can be a rigorous approach, but its generality is limited, and it requires great care with the side conditions (which are rarely spelled out); calculations of this sort can easily become confused and erroneous when networks and applications grow complex. Even when using *(7)* below, it is a good idea to test one's gradient calculations using explicit perturbations in order to be sure that there is no bug in one's code.

When the idea of backpropagation was first presented to the Harvard faculty in *1972,* they expressed legitimate concern about the validity of the rather complex calculations involved. To deal with this problem, I proved a new chain rule for ordered derivatives:

$$\frac{\partial^+ TARGET}{\partial z_i} = \frac{\partial TARGET}{\partial z_i} + \sum_{j>i} \frac{\partial^+ TARGET}{\partial z_j} * \frac{\partial z_j}{\partial z_i} \qquad (2.11)$$

Where the derivatives with the superscript represent ordered derivatives, and the derivatives without subscripts represent ordinary partial derivatives. This chain rule is valid only for ordered systems where the values to be calculated can be calculated one by one (if necessary) in the order $z_1, z_2, ..., z_n$, TARGET. The simple partial derivatives represent the direct impact of z_i on z_j through the system equation which determines z_j. The ordered derivative represents the total impact of z_i on TARGET, accounting for both the direct and *indirect effects.* For example, suppose that we had a simple system governed by the following two equations, in order:

$$z_2 = 4 * z_1$$

$$z_3 = 3 * z_1 + 5 * z_2$$

The "simple" partial derivative of z_3 with respect to z_1, (the direct effect) is *3;* to calculate the simple effect, we only look at the equation which determines z_3. However, the ordered derivative of z_3 with respect to z_1 is *23* because of the indirect impact by way of z_2. The simple partial derivative measures what happens when we increase z_1 (e.g., by 1, in this example) and assume that everything else (like z_2) in the equation which determines z_3 remains constant. The ordered derivative measures what happens when we increase z_1, and also recalculate all other quantities like z_2 which are later than z_1 in the causal ordering we impose on the system.

This chain rule provides a straightforward, plodding, "linear" recipe for how to calculate the derivatives of a given TARGET variable with respect to all of the inputs (and parameters) of an ordered differentiable system in only one pass through the system. This paper will not explain this chain rule in detail since lengthy tutorials have been published elsewhere [1], [11]. But there is one point worth noting: because we are calculating ordered derivatives of one target variable, we can use a simpler notation, a notation which works out to be easier to use in complex practical examples [11].

We can write the ordered derivative of the TARGET with respect to z_1 as "$F\_\ z_1$," which may be described as "the feedback to z_1 ". In basic backpropagation, the TARGET variable of Interest is the error E. This changes the appearance of our chain rule in that case to

$$F\_z_i = \frac{\partial E}{\partial z_i} + \sum_{j>1} F\_z_i * \frac{\partial z_j}{\partial z_i} \qquad (2.12)$$

For purposes of debugging, onecan calculate the true value of any ordered derivative simply by perturbing zj at the point in the program where z_i is calculated; this is particularly useful when applying backpropagation to a complex network of functions other than neural networks.

2.5.4 Concepts of Back Propagation in Time

Backpropagation through time-like basic backpropagation-is used most often in. pattern recognition today. In some applications-such as speech recognition or submarine detection, our classification at time t will be more accurate if we can account for what we saw at earlier times. Even though the training set still fits the same format as above, we want to use a more powerful class of networks to do the classification; we want the output of the network at time t to account for variables at earlier times.

The Introduction cited a number of examples where such "memory" of previous time periods is very important. For example, it is easier to recognize moving objects if our network accounts for changes in the scene from the time t -1 to time t, which requires memory of time t - 1. Many of the best pattern recognition algorithms involve a kind of "relaxation" approach where the representation of the world at time t is based on an adjustment of the representation at time t - 1; this requires memory of the internal network variables for time t - 1.

Backpropagation can be applied to any system with a well-defined order of calculations, even if those calculations depend on past calculations within the network itself. For the sake of generality, (2.7) will show how this works for the network design where every neuron is potentially allowed to input values from any of the neurons at the two previous time periods (including, of course, the input neurons). To avoid excess clutter, the hidden and output sections of the network (parallel to Fig.2.18) only for time T, but they are present at other times as well. To translate this network into a mathematical system, we can simply replace (2.8) above by

$$net_i(t) = \sum_{j=1}^{i-1} W_{ij} x_i(t) + \sum_{j=1}^{N+n} W'_{ij} x_j(t-1) + \sum_{j=1}^{N+n} W''_{ij} x_j(t-2) \qquad (2.13)$$

Again, we can simply fix some of the weights to be zero, if we so choose, in order to simplify the network. In most applications today, the *IN'* weights are fixed to zero (i.e., erased from all formulas), and all the W' weights are fixed to zero as well, except for Wi';. This is done in part for the sake of parsimony, and in part for historical reasons. (The "time-delay neural networks" of Watrous and Shastri [5] assumed that special case.) Here, I deliberately include extra terms for the sake of generality. I allow for the fact that all active neurons (neurons other than input neurons) can be allowed to input the outputs of any other neurons if there is a time lag in the connection. The weights Wand W" are the weights on those time-lagged connections between neurons. (Lags of more than two periods are also easy to manage; they are treated just as one would expect from seeing how we handle lag-two terms, as a special case of (2.12)). These equations could be embodied in a subroutine:

SUBROUTINE NET2 *(X(t), W', W'', x(t - 2), X(t-1), x(t), Yhat)*

which is programmed the subroutine NET, with the modifications one would expect from (15). The output arrays are *x(t)* and *Yhat*. When we call this subroutine for the first time, at t = 1, we face a minor technical problem: there is no value for x(-1) or x(0), both of which we need as inputs. In principle, we can use any values we wish to choose; the choice of x(-1) and 40) is essentially part of the definition of our network. Most people simply set these vectors to zero, and argue that their network will start out with a blank slate in classifying whatever dynamic pattern is at hand, both in the training set and in later applications. (Statisticians have been known to treat these vectors as weights, in effect, to be adapted along with the other weights in the network. This works fine in the training set, but opens up questions of what to do when one applies the network to new data.)

In this section, I will assume that the data run from an initial time t = 1 through to a final time t = T, which plays a crucial role in the derivative calculations.

To calculate the derivatives of F_W_{ij}, we use the same equations as before, except that (10) is replaced by

$$F\_x_i(t) = F\_\hat{Y}_{i-N}(t) + \sum_{j=i+1}^{N+n} W_{ji} * F\_net_j(t) + \sum_{j=m+1}^{N+n} W''_{ji} * F\_net_j(t+1) + \sum_{j=m+1}^{N+n} W''_{ji} * F\_net_j(t+2) \qquad (2.14)$$

Once again, if one wants to fix the W'' terms to zero, one can simply delete the right most term. Notice that this equation makes it impossible for us to calculate $F\_x_i(t)$ and $F\_net_i(t)$ until after $F\_net_j(t+1)$ and $F\_net_j(t+2)$ are already known; therefore, we can only use this equation by proceeding *backwards in* time, calculating *F_net* for time *T*, and then working our way backwards to time 1. To adapt this network, of course, we need to calculate F_W_{ij}, $F\_W'_{ij}$, and $F\_W''_{ij}$.

$$F\_W'_{ij} = \sum_{t=1}^{T} F\_net_i(t+1) * x_i(t) \qquad (2.15)$$

$$F\_W''_{ij} = \sum_{t=1}^{T} F\_net_i(t+2) * x_i(t) \qquad (2.16)$$

In all of these calculations, *F_net(T + 1)* and *F_net(T + 2)* should be treated as zero. For programming convenience, I will later define quantities $F\_net'_i(t) = F\_net_i(t+1)$, but this is purely a convenience; the subscript " *i* " and the time argument are enough to identify which derivative is being represented. (In other words, $net_i(t)$ represents a specific quantity Z_i as in (2.14), and $F\_net_i(t)$ represents the ordered derivative of E with respect to that quantity.)

2.5.5 Backpropagation Training for Fully Recurrent Nets

Backpropagation can be applied to a recurrent net with an arbitrary pattern of connections between the units. The process described here is the recurrent back propagation presented by Hertz, Krogh, and Palmer (1991)[18], based on work by Pineda [1987, 1988, 1989] and Almeida [1987, 1988].The activations of our general recurrent net are assumed to obey the evolution equations of Cohen and Grossberg (1983) and Hopfield (1984) [15], namely,

$$\gamma \frac{dv_i}{dt} = -v_i + g(x_i + \sum_j v_j w_{ij}) \ ,$$

where x_i is the external input to unit v_i and γ is a time constant. We assume that at least one stable attractor exists, i.e.,

$$v_i = g(x_i + \sum_j v_j w_{ij}) \ ,$$

To train a recurrent net using backpropagation, we assume that target values are specified for some units, which we call output units. The error is defined in the usual manner for these units, i.e.,

$$E = \frac{1}{2} \sum_k (t_k - v_k)^2 \ ,$$

where the summation ranges over all the output units[9]

Gradient descent applied to this net gives a weight update rule that requires a matrix inversion at each step. However, if we write the weight updates as

$$\Delta w_{pq} = \alpha \delta_q v_p$$

where

$$\delta_q = g'(x_q + \sum_j v_j w_{qj}) y_q$$

(in which the matrix inversion is included in the term *Yq),* we find [Hertz, Krogh, & Palmer, 19911 that the *yq* terms obey a differential equation of exactly the same form as the evolution equations for the original network. Thus, the training of the network can be described by the following algorithm:

Step 1. Allow the net to relax to find the activations vi; i.e., solve the equation

$$\gamma \frac{dv_i}{dt} = -v_i + g(x_i + \sum_j v_j w_{ij})$$

Define the equilibrium net input value for unit *q:*

$$h_q = (x_q + \sum_j v_j w_{qj})$$

Step 2. Determine the errors for the output units, *E,*

Step 3. Allow the net to relax to find the y_q; i.e., solve the equation

$$\gamma \frac{dy}{dt} = -y_q + E_q + \sum_k g'(h_k) w_{qk} y_k$$

The weight connections of the original net have been replaced by $g'(h_k) w_{qk} y_k$ and the activation function is now the identity function.

The error term, E_q, plays the role of the external input.

Step 4. Update the weights:

$$\Delta w_{pq} = \alpha v_p g'(h_q) y_q,$$

where v_p is the equilibrium value of unit p, y_q is the equilibrium value of the "matrix inverse unit," and h_q is the equilibrium net input to unit q.

2.6 Applications of Different Types Neural Networks

It is important to know that there are different types of networks, as here the type of Recurrent Multilayered Neural Network and the applications work with Back Propagation in time will be discussed. These applications are very important for the implementation of this work in the mordern industry [20].

2.6.1 Application of Recurrent multilayered Neural network

One example of the use of a simple recurrent net demonstrates the net's ability to learn an unlimited number of sequences of varying length [Servan-Schreiber, Cleeremans, & McClelland, 1989]. The net was trained to predict the next letter in a string of characters. The strings were generated by a small finite-state grammar in which each letter appears twice, followed by a different character. A diagram of the grammar is given in Figure 2.19. The string begins with the symbol B and ends with the symbol ε.

At each decision point, either path can be taken with equal probability. Two examples of the shortest possible strings generated by this grammar are

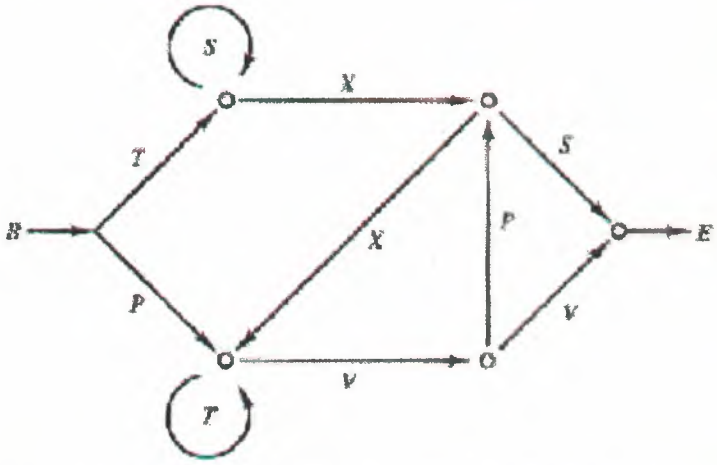


Figure 2.19 One grammar for simple recurrent net.

B P V V E
and
B T X S E

The learning patterns for the neural net consisted of 60,000 randomly generated strings ranging in length from 3 to 30 letters (not including the Begin and End symbols).

The neural net architecture for this example had six input units (one for each of the five characters, plus one for the Begin symbol) and six output units (one for each of the five characters, plus one for the End symbol). There were three hidden units (and therefore, three context units). In a specific case, the net might be displayed as in Figure 2.20. With the architecture as illustrated, the input pattern for the letter B (the Begin symbol) would correspond to the vector (1, 0, 0, 0, 0, 0).

Training the net for a particular string involves several steps, the number depending on the length of the string. At the beginning of training, the activations of the context units are set to 0.5. The first symbol (the Begin symbol) is presented to the input units, and the net predicts the successor. The error (the difference between the predicted and the actual successor specified by the string) is determined and backpropagated, and the weights are adjusted. The context units receive a copy of the hidden units activations, and the next symbol in the string (which was the target value for the output units on the first step of training) is presented to the input units. Training continues in this manner until the End symbol is reached.

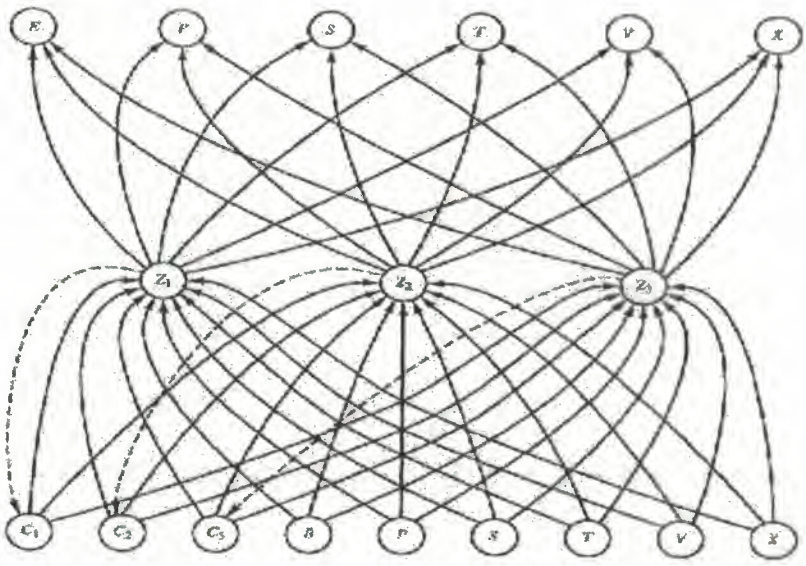


Figure 2.20 Simple recurrent net to learn context-sensitive grammar.

The training algorithm for a context-sensitive grammar in the example given is as follows:
For each training string, do Steps 1-7.

Step 1. Set activations of context units to 0.5.
Step 2. Do Steps 3-7 until end of string.
Step 3. Present input symbol.
Step 4. Present successor to output units as target response.
Step 5. Calculate predicted successor.
Step 6. Determine error, backpropagate, update weights.
Step 7. Test for stopping condition:
If target = E, then
stop;
otherwise,
Copy activations of hidden units to context units;
continue.

As a specific example of the training process, suppose the string

B T X S E

is used for training. Then we have:

Step 2. Begin training for this string.
- Step 3. Input symbol B, i.e., (1, 0, 0, 0, 0, 0).
- Step 4. Target response is T, i.e., (0, 0, 0, 1, 0, 0).
- Step 5 Compute predicted response, a real-valued vector with components between 0 and 1.
- Step 6 Determine error, backpropagate, update weights.
- Step 7 Copy activations of hidden units to context units.

Step 2. Training for second character of the string.
- Step-3. Input symbol T, i.e., (0, 0, 0, 1, 0, 0).
- Step 4. Target response is X, i.e., (0, 0, 0, 0, 0, 1).
- Step 5. Compute predicted response,
- Step 6. Determine error, backpropagate, update weights.
- Step 7. Copy activations of hidden units to context units.

Step 2. Training for third character of the string.
- Step 3. Input symbol X, i.e., (0, 0, 0, 0, 0, 1).
- Step 4. Target response is S, i.e., (0, 1, 0, 0, 0, 0).
- Step 5-7. Train net and update activations of context units.

Step 2. Training for fourth character of the string.
Step 3. Input symbol S, i.e., (0, 1, 0, 0, 0, 0).
Step 4. Target response is E, i.e., (1, 0, 0, 0, 0, 0).
Steps 5-6. Train net.
Step 7. Target response is the End symbol; training for this string is complete.

After training, the net can be used to determine whether a string is a valid string, according to the grammar. As each symbol is presented, the net predicts the possible valid successors of that symbol. Any output unit with an activation of 0.3 or greater indicates that the letter it represents is a valid successor to the current input. To determine whether a string is valid, the letters are presented to the net sequentially, as long as the net predicts valid successors in the string. If the net fails to predict a successor, the string is rejected. If all successors are predicted, the string is accepted as valid.

The reported results for 70,000 random strings, 0.3% of which were valid according to the grammar, are that the net correctly rejected all of the 99.7% of the strings that were invalid and accepted all of the valid strings. The net also performed perfectly on 20,000 strings from the grammar and on a set of extremely long strings (100 or more characters in length).

2.6.2Application of Back Propagation in Time

A neural network with no hidden units has been trained to act as a simple shift register using backpropagation in time [Rumethart, Hinton, & Williams, 1986a]. For example, consider the network shown in Figure 2.21, with three input units and three output units. (In practice, these units can be the same, but we will treat them as distinct to emphasize the similarities with Figures 2.16 and 2.18. For simplicity, the weights are not shown in the figure or in the diagrams that follow. In addition to the units shown, each unit receives a bias signal.

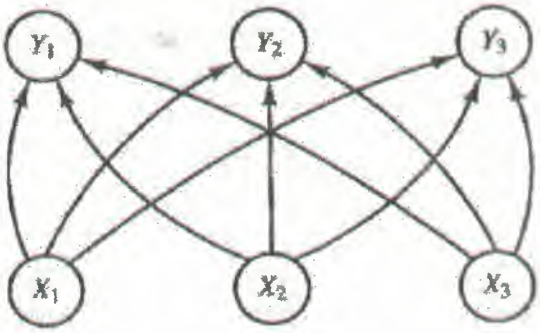


Fig. 2.21 Recurrent net used as shaft register.

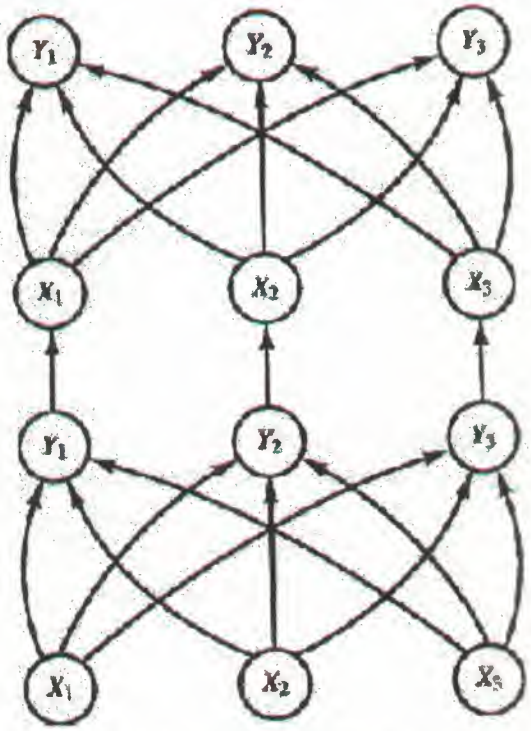


Fig 2.22 Expanded diagram of Recurrent net used as shift Registers.

The training patterns consist of all binary vectors with three components; the target associated with each vector is the pattern shifted two positions to the left (with wraparound). This is the desired response of the net after two time steps of processing. The expanded form of the net is shown in Figure 2.22. This example illustrates the fact that it is not required to have information on errors at the intermediate time steps. If the net were told the desired response after one time step, the solution would be very simple. Instead, the weights in both copies of the net are adjusted on the basis of errors after two time steps. In general, a combination of information on errors at the final level and at any or all intermediate levels may be used. Rumelhart, Hinton, and Williams (1986a, 1986b) found that the net consistently learned the weights required for a shift register in 200 or fewer epochs of training, with a learning rate of 0.25, as long as the bias weights were constrained negative. The same conclusions apply to the net with five input (and output) units. In either of these cases, if the biases are not restricted to be negative, other solutions to the training can also result. These give the desired results after an even number of time steps, but not after an odd number of time steps.

Chapter 3
IMPLEMENTATION OF NEURAL NETWORK FOR TECHNOLOGICAL PROCESS

3.1. Introduction

A recurrent multilayer Perceptron (RMLP) model is designed and developed for simulation of core neutronic phenomena in a technological process plant, which constitute a non-linear, complex dynamic system characterized by a large number of state variables. A modified back propagation learning algorithm with an adaptive steepness factor is employed to speed up the training of the RMLP. The test results presented exhibit the capability of the recurrent neural network model to capture the complex dynamics of the system, yielding accurate predictions of the system response. The performance of the network is also demonstrated for interpolation, extrapolation, fault tolerance due to incomplete data, and for operation in the presence of noise.

In recent years, there has been considerable interest in modeling dynamic systems with neural networks. The basic motivation is the ability of neural networks to create data driven representations of the underlying dynamics with less reliance on accurate mathematical/physical modeling. There exist many problems for which such data-driven representations offer advantages over more traditional modeling techniques, such as availability of fast hardware implementations, ability to cope with noisy/incomplete data, and avoidance of complicated internal state representations.

In this work, we make a system identification of core neutronic phenomena in a nuclear reactor using a Recurrent Multilayer Perceptron (RMLP). The training data is obtained from REMARK, a first principles neutronic core model which can model both normal operation and fast and localized transients. REMARK in its plant specific versions is currently being used to train personnel who will be working in electric power plants and is tested with a number of standard real plant data to verify its approximation capability. We can simulate various scenarios in nuclear plant operation such as reactor and pump trips and generate training and test data sets which obviously can not easily be obtained in a real nuclear plant.

We introduce a novel modified backpropagation scheme for training the RMLP to speed up learning. The proposed scheme keeps the network weights active in early stages of learning by adapting steepness of the activation function and using an annealing schedule to ensure convergence. It is demonstrated that, by the procedure, learning speed is improved considerably, a feat especially important in a modeling task of this size which consists of some 146 inputs and 52 outputs.

The performance of RMLP model is evaluated in terms of its prediction performance on unseen test sets for interpolation and extrapolation, fault tolerance in cases of incomplete data, and for operation in the presence of additive white gaussian noise. Accurate predictions provided by RMLP demonstrate its success in capturing the underlying complex dynamics of the neutronic core.

The specific advantages of RMLP for core electronic modeling are:

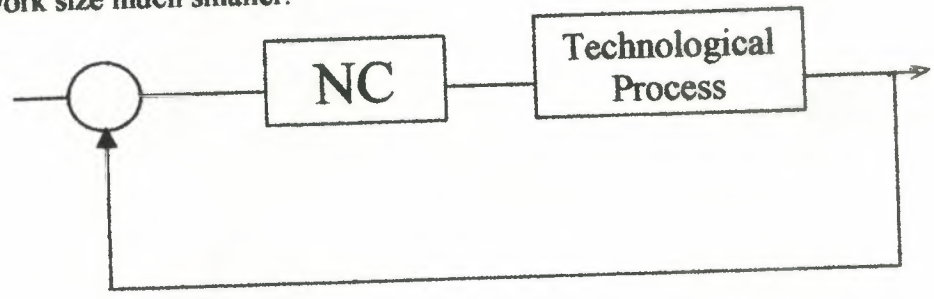
(i) Providing speedups in system prediction by using dedicated hardware which provide much needed faster than real-time prediction power;

(ii) Better robustness to noisy data, ability to provide estimates of the system response in the presence of missing measurements (possibly due to malfunctioning equipment) which are *harder to* deal with in a model based implementation;

(iii) The capability to start predictions from arbitrary initial conditions which allow recovery of transients without the need to set internal parameters necessary in model based implementations.

3.2. The Recurrent Multilayer Perceptron (RMLP) Model for Technological Processes Control

We employ a recurrent network structure to capture the dynamics of the neural controller for technological process core as represented by transients under different operating conditions. We adopt a global RMLP which incorporates a unit delayed connection between the output layer and the input layer. The structure is illustrated in Fig.3.1. Reference [8] presents an experimental comparative study of various recurrent architectures on different classes of problems and reports that the RMLP and Narendra and Parthasarathy's [10] model outperform other recurrent architectures that they consider for a nonlinear system identification problem. In [1], we show that the RMLP structure provides a powerful representation of the core dynamics outperforming a time delay neural network (TDNN) structure in that it provides a much better approximation of the input dynamics with a network size much smaller.



3.1 Stucture of Neural Network For Technological Process.

Introductroduction of the notation for the globally recurrent MLP as follows: Let **x**(n) denote the N x 1 external input vector applied to the network at time n, and let **y**(n) denote the corresponding M x 1 vector of outputs at time n. The external input vector **x**(n) and one step delayed output vector **y**(n - 1) are concatenated to form the input layer to the network. The network is fully interconnected with **V**(n) and **W**(n) as its weight matrices.

Let $v_{ij}(n)$ denote the connection weight between *i*th input node and *j*th hidden neuron, and $w_{ij}(n)$ denote the connection weight between *i*th hidden neuron and *j*th output neuron. The output of *k*th output neuron at time n is given by:

$$fv,w : R^{N+M} \rightarrow R^{M} \qquad (3.1)$$

and we define the *k*th neuron output as

$$y_k(n) = [fv,w(x(n),y(n-1)]_k = \sigma(s_k(n)) \qquad (3.2)$$

with

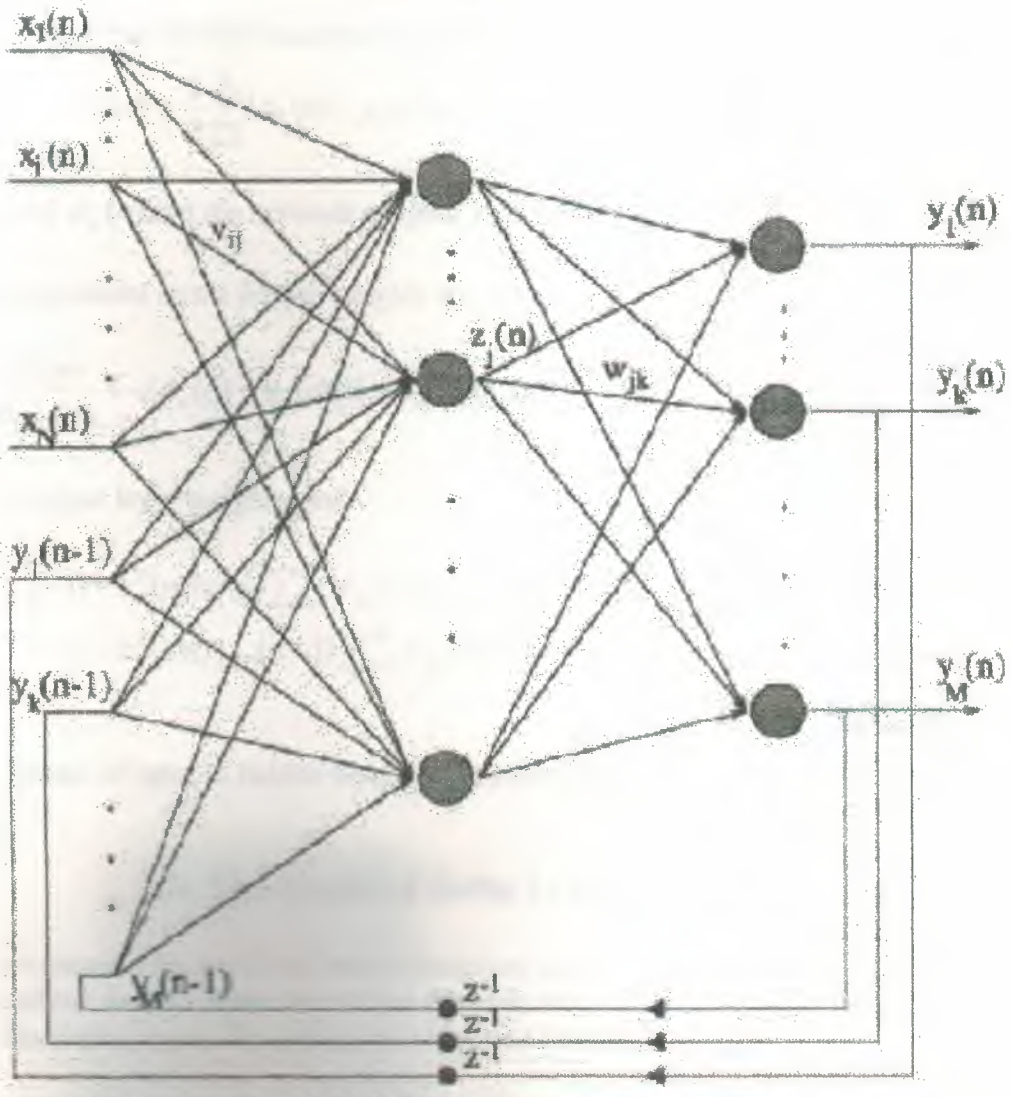


Fig 3.2 The stucuture of RMLP (Recurrent multi layered Perceptron).

$$s_k(n) = \sum_{j=1}^{J} w_{jk}(n) z_j(n) \qquad (3.3)$$

and

$$z_k(n) = \sigma\left(\sum_{i=1}^{N} v_{ij}(n) x_i(n) + \sum_{m=1}^{M} v_{(N+m)k}(n) y_m(n-1) \right) \qquad (3.4)$$

where J is the number of neurons in the hidden layer, $\sigma(.)$ is the node activation function, and $k = 1,...,M$.

With the notation introduced above, the weights w_{jk} and v_{ijs} are adapted by the steepest descent rule

$$w_{jk}(n+1) = w_{jk}(n) - \mu \frac{\partial E(n)}{\partial w_{jk}(n)} \qquad (3.5)$$

(similarly for v_{ij}) such that the instantaneous squared error

$$E(n) = \frac{1}{2}\sum_{k=1}^{K}(d_k(n) - y_k(n))^2 \qquad (3.6)$$

between the desired $d_k(n)$ and the network outputs $y_k(n)$ is minimized over all outputs.

The instantaneous gradient terms for the weights are given by

$$\frac{\partial E(n)}{\partial w_{jk}(n)} = -(d_k(n) - y_k(n))(1 - y_k^2(n))z_j(n) \qquad (3.7)$$

for the hidden to output layer weights, and

$$\frac{\partial E(n)}{\partial v_{ij}(n)} = \begin{cases} -(1 - z_j^2(n))x_i(n)\sum_{k=1}^{M} w_{jk}(n)(d_k(n) - y_k(n))(1 - y_k^2(n)) \ldots\ldots\ldots i \le N \\ -(1 - z_j^2(n))y_{i-N}(n-1)\sum_{k=1}^{M} w_{jk}(n)(d_k(n) - y_k(n))(1 - y_k^2(n)) \ldots N \le i \le N+M \end{cases} \qquad (3.8)$$

for the update of input to hidden layer weights through backpropagation of the error signal.

3.3. The Modified Delta Learning Rule

In this section, we introduce a modified backpropagation scheme for training the RMLP. The algorithm is a gradient descent technique with a dynamic steepness factor in the activation functions of the nodes in the hidden layer. We define the activation functions as

$$\sigma(\sigma_j) = \tanh(\lambda \alpha_j) \qquad (3.9)$$

where λ is the steepness factor, and α_j is the total input of the jth hidden neuron such that

$$\alpha_j = \sum_{i=1}^{N} v_{ij}(n)x_i(n) + \sum_{m=1}^{M} v_{(N+m)j}(n)y_m(n-1) \qquad j = 1, \ldots, J \qquad (3.10)$$

Convergence of backpropagation-type learning algorithms in neural networks employing sigmoid-like activation functions suffer from premature increases in some node weights which force node outputs to the saturated region of the activation function in early stages of learning. The learning rate of the nodes is thus diminished due to vanishingly small gradients. There have been several proposed fixes for this problem in the literature, such as independent learning rates for each weight, adaptation of learning rates during learning [5, 9, 11], scaling [13] or adding a constant [6] to the the gradient term backpropagated to the previous layers. We propose the following technique for adaptation of the steepness factor A of the activation functions in the hidden layer depending on the magnitude of the inputs to the hidden layer node. The goal is to keep the weights active during

the early stages of learning by adjusting steepness according to a representative node selected by the maximum. magnitude of the input to hidden node. To ensure convergence, an annealing schedule which, as training progresses, decreases the steepness adjustment is also proposed. The resulting algorithm takes aggressive steps early due to the large gradients and converges as the steepness adjustment is cooled down.

We can write the adaptive procedure for the steepness of the activation function as

$$\lambda(n) = \frac{\lambda_0}{\max\{\alpha_1, \alpha_2, \ldots, \alpha_J\}} p(n) + \lambda_1 (1 - p(n)) \qquad (3.11)$$

where $p(n)$ is the annealing parameter which can be chosen such that $p(1) = 1$ and $p(n) \to 0$ as $n \to \infty$. A typical choice for $p(n)$ is a sequence of order $O(n^{-\alpha})$ for some $a > 0$. We demonstrate the effect of annealing in the update with a simple example. Consider the non linear system identification problem where the system to be identified is described by the relationship,

$$d(n) = \frac{x(n)(1 + d(n-1)}{1 + d^2(n-1)} \qquad (3.12)$$

and the input is chosen as a monotonously decreasing ramp in the range [-1,1]. A 2-3-1 globally RMLP is trained with 80 training samples. In Fig. 3.2 we show the initial speed up in the learning with the steepness update with $(p(n) = 1/n)$ and without annealing $(p(n) = 1)$ for 50 epochs. The adaptive steepness learning curves for both cases (with and without annealing) show a considerable speed up in convergence. Also, with annealing, the network achieves a much lower final training error. Fig. 3 indicates a similar trade-off for the same system identification problem (12) where this time λ_0 is let to change with no annealing.

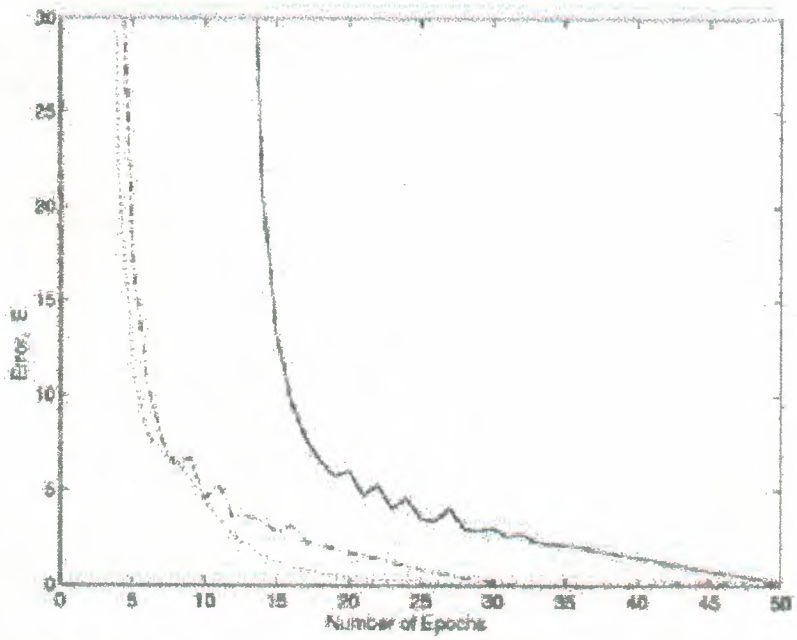


Fig3.3. The effects of steepness and annealing procedure for system indentification problem. Solid standard back propogation, dotted: steepness adaptation with annealing process.

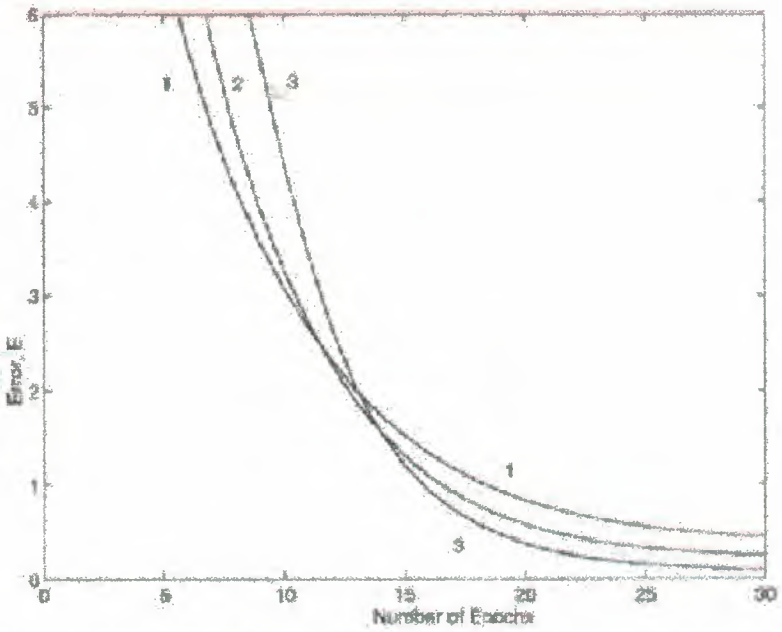


Fig 3.4: The effects of λ_0 in steepness adaptationfor the simple system identification problem.

In the nuclear reactor core modeling problem, the modification of backpropagation algorithm results in considerable improvement in terms of both the learning characteristics and accuracy. This improvement is illustrated in Fig. 3.5 by comparing the two learning curves, that are obtained with (dotted line) and without (solid line) the adaptation of the activation function. Since there is a natural convergence in the value of $\lambda(n)$, we do not employ annealing for this problem, hence *p(n) is* set to 1. The value of λ_0 is chosen as the steepness value $\lambda(n)$ naturally converges to, which is 6 in our implementation. The behavior of the adaptive steepness factor $\lambda(n)$ for our reactor core problem is shown in Fig.3.6. With the adaptation, the number of iterations required to reach the same training error is much smaller than the number of iterations required without the adaptation. The final value of the training error is also reduced considerably.

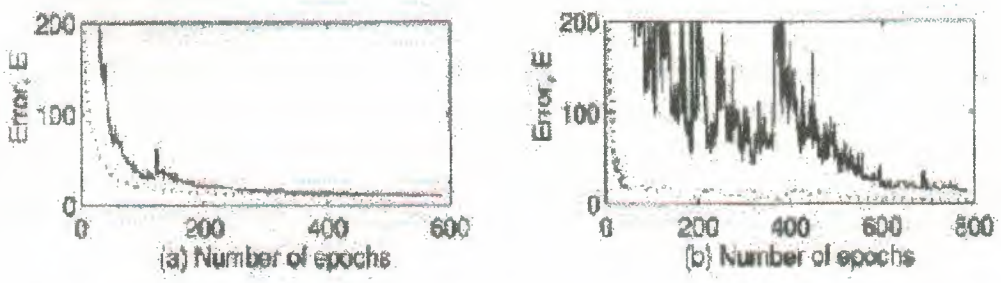


Fig 3.5 Effects of steepness adaptation for reactor core modelling. (a) Learning curves for single trip scenario, (b) Learning curves for two pump trip scenario.

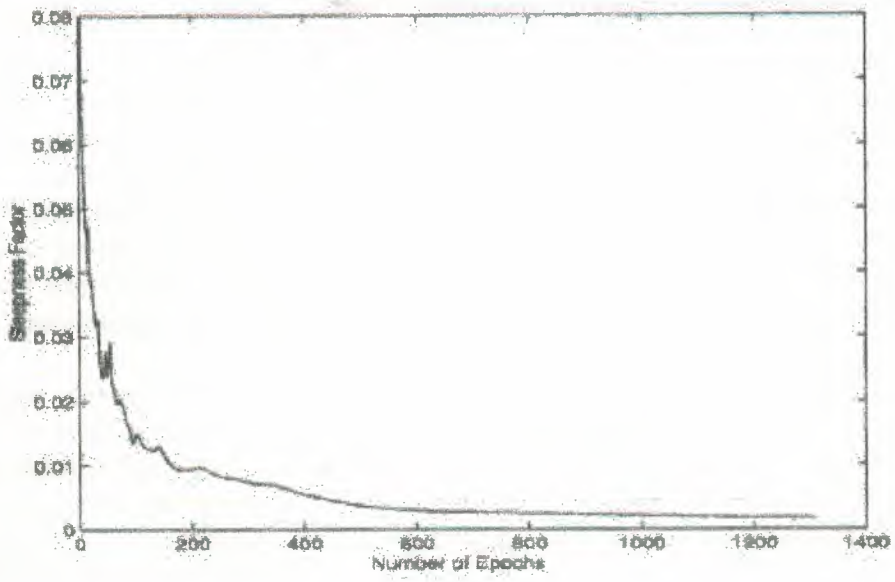


Fig 3.6 Adaptive steepness λ (n) during training for the network core modelling for two pump trip scenario.

3.4. Implementation of the Neural Controller

The version of REMARK we use has 146 external inputs and 52 external outputs that characterize the reactor core physics, and internal state variables as discussed in the appendix. To generate training and test data sets, REMARK is run with various scenarios, which describe different operation conditions of the technological process plant. All of the scenarios used in our experiments assume a disturbance of the system from its steady state operating condition, i.e., modeling of various transient responses of the system. These transients are achieved by setting the internal parameters of REMARK. The steady state response of the reactor core as we show in [3], can be achieved by a simple feedforward structure. In [3], we present a full model of the system (the steady state and the three types of transient responses considered) using a switch mechanism between three RMLPs and a simple feedforward MLP. The three transient scenarios are core trip, single pump trip, and double pump trips. In our implementation, the switch uses REMARK internal variables for control of the switching mechanism. In a real plant scenario, various plant parameters can be used for this purpose, such as using the rod positions for checking for reactor trip [3]. The structure of the switch mechanism is excluded in this paper because of the additional introduction it necessitates to the structure of REMARK.

Due to the configuration of the circulation pumps around the reactor core tripping different combination of circulation pumps (for the double pump trip scenario) and tripping a different pump (for the single pump trip scenario) result in different transient characteristics. When the pump trips are coupled with a reactor shut down, rapid transients occur with reactor shut down having the dominant effect. For pump trip(s) without reactor shut down, we observe slower transient characteristics. For single pump trips, the power distribution in the reactor core varies depending on the location of the pump tripped. Therefore, to evaluate the generalization performance of the network we use data generated by tripping a different pump than the one used in training. For double pump trips, we consider tripping two pumps on the opposing sides of the cooling system

since this scenario does not cause system unbalance due to reverse leakage of coolant fluid as the parallel pump trips do. Again for testing we use data collected by tripping a different two-pump combination than the one used in training. We collect samples generated by REMARK with 4 Hz. sampling frequency. The network is trained by using the first 1000-1500 samples (samples 1/4-sec. apart) and the generalization performance is tested on a different data set (obtained by tripping a different pump or combination of pumps). We also test for samples after the initial n samples used in training to see the prediction capability of the network into the future. Also, of particular interest is starting the network from an arbitrary (in our case all zeros) initial condition instead of its steady-state value to see the ability of the network to generalize to this case. The performance of the network in this test (start-up from an arbitrary initial condition) presents a particular advantage of the neural network modeling with respect to first principles modeling as will be explained later.

Through proper setting of the internal variables, REMARK can generate real-time data corresponding to various scenarios, which describe different operation conditions of the nuclear plant. Due to the configuration of the circulation pumps around the reactor core, tripping different combination of circulation pumps with or without a total controller shut down result in different transient characteristics. The training and testing data for the RMLP are obtained through various combinations of such operating scenarios.

CONCLUSION

The graduation project is devoted to developing Intellectual Systems for Technological Processes Control. To solve given problems, the state of art of understanding Intellectual Systems for solving control problems is described. The developing of Intellectual Control Systems on the base of Recurrent Neural Networks is shown.

Following the above, there are a detailed descriptions of the subjects related to the Neural Technology. These subjects mainly include the structure and learning of algorithms of the Neural Networks. The project also describes the models of different neurons, structures, architectures of Recurrent Neural Networks, supervised learning problems, concepts and theories of Back Propagation and also the applications that relate them.

Learning algorithms of Non-Recurrent and Recurrent Neural Networks are also explained in detail.

Besides Recurrent Neural Networks, we also discuss a very important field, which handles most of the implementation and application problems of Neural Networks, which is its learning.

More used algorithms for Neural Networks learning is Back Propagation. In the project, different versions of Back Propagation, algorithms, structures, derivatives, and applications are considered. There are different types of architectures of Neural Networks, which are described in the text as well. The application of Back Propagation, algorithm for this networks which is necessary to be considered is given.

The modeling of control systems for Technological Processes, based on Recurrent Multilayered Artificial Neural Networks is given. Using Back Propagation Algorithm, the synthesis (training) process of Intellectual Control System for Technological Processes is considered.

The suggested approach allows us to increase the efficiency of control systems.

Taking into account, described concepts and theories, the implementation for Intellectual Systems is realized and the results and simulations are discussed.

REFERENCES

1. P.Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Committee on Appl. Math., Harvard Univ., Cambridge, MA, Nov. 1974.
2. D. Rumelhart, D. Hinton, And G. Williams, "Learning internal representations by error propagation," in D. Rumelhart and F. McClelland, eds., Parallel *Distributed Processing,* Vol. 1. Cambridge, MA: M.I.T. Press, 1986.
3. D.B. Parker, "Learn i ng-logic," M.I.T. Cen. Computational Res. Economics Management Sci., Cambridge, MA, TR47,1985.
4. Y. Le Cun, "Une procedure d'apprentissage pour reseau a seuil assymetrique," in Proc. Cognitiva85, Paris, France, June 1985, pp. 599-604.
5. R. Watrous and L. Shastri, "Learning phonetic features using connectionist networks: an experiment in speech recognition," in Proc. 1st *IEEE* Int. Cont. Neural Networks, June 1987.
6. H. Sawai, A. Waibel, P. Haff ner, M. Miyatake,and K. Shikano, "Parallelism, hierarchy, scaling in time-delay neural networks for spotting Japanese phonemes/CV-syl lab les, " in Proc. *IEEE* Int. Joint *Conf.* Neural *Networks, June* 1989.
7. D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," in W. T. Miller, R. Sutton, and P. Werbos, Eds., Neural Networks for *Robotics* andControl. Cambridge, MA: M.I.T. Press, 1990.
8. M. Jordan, "Generic constraints on underspecified target trajectories," in Proc. *IEEE* Int. Joint Conf. Neural *Networks, June* 1989.
9. M. Kawato, "Computational schemes and neural network models for formation and control of multijoint arm trajectory," in W. T. Miller, R. Sutton, and P. Werbos, Eds., Neural *Networks* for *Robotics and Control.* Cambridge, MA: M.I.T. Press, 1990.
10. R. Narendra, "Adaptive control using neural networks," in W. T. Miller, R. Sutton, and P. Werbos, Eds., Neural *Networks for Robotics and Con trol.* Cambridge, MA: M.I.T. Press, 1990.
11. P.Werbos, "Maximizing long-term gas industry profits in two minutes in Lotus using neural network methods," *IEEE Trans. Syst., Man, Cybern., Mar./Apr.* 1989.

12. "Backpropagation: Past and future," in *Proc. 2nd IEEL Int.* Conf. Neural *Networks, June* 1988. The transcript of the talk and slides, available from the author, are more introductory in nature and more comprehensive in some respects.

13. Guyon, 1. Poujaud, L. Personnaz, G. Dreyfus, J. Denker, and Y. Le Cun, "Comparing different neural network architectures for classifying handwritten digits," in Proc. *IEEE* Int. joint Conf. Neural Networks, June 1989.

14. P. Werbos, "Applications of advances in nonlinear sensitivity analysis," in R. Drenick and F. Kozin, Eds., Systems *Modeling and* Optimization: Proc. 10th IFIP Cont. (1981). New York: Springer-Verlag, 1982.

15. "Generalization of backpropagation with application to j a recurrent gas market model," Neural Networks, Oct. 1988.

16. R.A.Aliev, R.H.Abiev, R.R.Aliev. Industrial controllers based on neural expert system.//World congress on Expert systems, Estoril/Lissabon, Portugal,1994.

17. R.H.Abiev, K.W.Bonfig, F.T.Aliev. Controller based on fuzzy neural network for control of technological process.// International conferences on Application of Fuzzy Systems and Softcomputing, Siegen, Germany, 1996, June 25-27.

18. R.A.Aliev,F.T.Aliev,R.H.Abiev,R.R.Aliev.Industrial neural controllers.// EUFIT'94, Elita foundation, Promenade 9, 52076 Aachen, Germany,1994.