A Neural Framework For Recognizing Time-Varying Visual Signals

by

Tet Hin Yeap

A Thesis
Submitted in Conformity
with the Requirements
for the Degree of
Doctor of Philosophy
in the
University of Toronto

June 1992

Department of Electrical Engineering
University of Toronto
Toronto, Ontario, CANADA

Tet Hin Yeap

University of Toronto

Department of Electrical Engineering

A Neural Framework For Recognizing Time-Varying Visual Signals

Abstract

This thesis proposes a modular architecture for neural networks that performs temporal pattern recognition. The model proposes a special configuration of a sequential neural network with one memory element per state to keep track of individual events in a sequence. Temporal integration of the network state that represents the recognition result of each event is used to generate an overall response representing the sequence. Modules can be concatenated to recognize a long temporal sequence. When used in a multilevel hierarchy, modules at higher levels can be trained to recognize the temporal order of sequences presented to modules in the lower levels.

The network behaviour has been studied both analytically and through simulation. Analytical results yield an estimate of the network parameters for a given application.

A motion detector based on the proposed network is presented to illustrate the capabilities of the proposed structure. It can be trained to recognize the motion of an object with different speeds along any trajectory. Simulation results show that the detector is insensitive to small differences between the training and test trajectories, can recognize partial sequences, and is tolerant to noise.

The proposed structure uses a small number of interconnections between modules. Hence, it is well suited for implementation in the form of VLSI chips. Simple circuits are presented for this purpose.

Acknowledgements

I would like to thank my thesis supervisors, Dr. S.G. Zaky, Dr. J.K. Tsotsos, and Dr. H.C. Kwan for their advice, wisdom and assistance. To be supervised by them was a stimulating and rewarding experience. They gave me the freedom to work on my own, yet they were always there to assist and guide me when I was lost.

I must express my gratitude to my family for their support and encouragement in this endeavor. I must also extend my gratitude to the rest of the staff and students of the Computer group of the Electrical Engineering Department, Computer Vision group of the Computer Science Department, and the Physiology Laboratory of the Medical Science Department.

I am indebted to the Graduate School of the University of Toronto for its support in the form of an Open Doctoral Fellowship.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1. Introduction

In temporal pattern recognition problems, such as motion detection or speech recognition, the information to be processed consists of a sequence of temporally related events. Unlike static pattern recognition where patterns are processed independently of each other, the task of temporal pattern recognition involves recognizing all the patterns in a sequence, and accounting for the order in which they occur and the time delay between successive patterns. In general, it requires a massive amount of computation.

This thesis presents a new structure for temporal pattern recognition in real time using a sequential neural network, that is, one whose next state is determined both by the network's current state and current input. The network generates a single response representing a sequence of events by utilizing the process of temporal integration. The response is generated in small increments at each time step by summing in time the recognition result of each event. The temporal integration process is made possible, and the mathematical analysis of the network is simplified, by using one memory element per state in the network's implementation.

The combined use of one memory element per state and the temporal integration process yields a network that is modular, easy to train, tolerant to noise, and can recognize partial sequences.

1.2. Temporal Pattern Recognition

We generally experience events as continuous functions of time. Input signals impinging on the sensory organs are time varying. For example, the perception of an aeroplane moving across the sky is a scene that is constantly changing. This is due to both the motion of the plane and the motion of the observer.

In many physical systems that process time-varying information, time is quantized into discrete steps. Inputs are sampled and held at fixed time intervals, generating a sequence of temporally related events [Tsotsos 1987, Mozer 1988, Lippmann 1989]. The task of temporal pattern recognition in such systems involves processing information which is presented in the form of one event per time step. After a sufficient number of events in a sequence have been presented, the system may be required to generate a response representing the sequence.

Temporal pattern recognition is often found in problems such as motion detection and speech processing [Aggarwal 1986, Lippmann 1989]. In motion detection, a sequence of images is processed and information about the motion of an object is generated. Similarly, in speech recognition, spoken words are recognized from acoustic signals sampled during the period of utterance.

Unlike a static pattern recognition system where input patterns are treated one at a time independent of each other, a temporal pattern recognition system needs to take into account the temporal order and the elapsed time between patterns, because each pattern is temporally related to the patterns before and after. The temporal pattern recognition system considered in this thesis has several important features based on those described in [Tsotsos 1980, Tsotsos 1987]. These features relate to the way in which the time element is treated during the computation process. They may be summarized as follows:

- Spatial information is presented and processed in parallel.
- Temporal information is presented and processed serially.
- The system performs the recognition process in real time.

- Two identical inputs at different time steps are recognized as different, depending on what patterns precede or follow them.
- The system generates meaningful output for a partial sequence.
- A correct sequence may be embedded into other sequences or noise, and may start at any time.
- The system is stable in a noisy environment, and degrades gracefully with increasing noise.

1.3. Thesis Motivation

The thesis has been motivated by the need for a system that is capable of producing a response in real time and by the conjecture that this task is realizable in the form of a neural network.

Neural networks have attracted wide-spread research interest because they are able to perform computations in parallel, they are fault tolerant, and most importantly, they can be trained to recognize different patterns [Hopfield 1982, Ballard 1984, Hinton 1984, Kohonen 1984, Hopfield 1986, Rumelhart 1986, Hinton 1989, Lippmann 1989]. A neural network is a set of nodes interconnected by weighted connections. It is trained to recognize a set of patterns by adjusting the weights of these connections using certain learning rules [Hinton 1984, Ackley 1985, Rumelhart 1986, Rumelhart 1986b, Hinton 1989, Lippmann 1989].

Recent work on neural networks has concentrated on the processing of static patterns, which consist of a set of input values presented to the network in parallel. Such networks have no means for representing time or for remembering previous input patterns. Hence, they are not suitable for temporal pattern recognition.

Two approaches to temporal pattern recognition based on neural networks have been proposed and will be described in chapter 2. They are time-delay networks [Lang 1988, Widrow 1988, Lippmann 1989] and sequential networks [Jordan 1986, Prager 1986, Shamma 1989]. These approaches have limitations. This thesis proposes the use of a special case of a sequential network,

which will be presented in chapter 3, to overcome some of these limitations.

A child can catch a ball with ease. However, before the child can catch the ball, it needs to recognize the trajectory and then estimate the next position of the ball. The brain is obviously able to process time-varying inputs effortlessly and naturally. The temporal behavior of a neuron, Barlow's early vision model for rabbits, and Tsotsos's high-level vision model have influenced the approach taken in this thesis [Barlow 1964, Barlow 1965, Kandel 1985, Tsotsos 1987].

1.4. Thesis Objectives

Given the following design specifications for sequence:

T – sampling period

T_{total} – duration of interest

N_{off} – maximum number of events for which the recognition result is negative

N_{on} – minimum number of events for which the recognition result is positive

the main objective of this thesis is to design a neural network which can be trained to recognize a temporal sequence, detect partial sequences, and is stable in a noisy environment. It will be shown in this thesis that a special form of a sequential neural network, together with an integrator, can be used to achieve this objective and meet all the design constraints mentioned in the previous section.

Most neural networks for temporal pattern recognition were designed and tested based on computer simulation. Since simulation alone is not sufficient to provide a full understanding of the dynamics of a network, the second objective of the thesis is to study the behavior of the proposed network using both simulation and mathematical analysis. The analysis provides means for estimation of the network parameters.

The VLSI (Very Large Scale Integration) implementation of a large neural network on one chip is technically infeasible, due to the limited number of external connections and transistors that

can be fabricated on a die [Mead 1980, Mead 1989]. One way to overcome these limitations is to partition a network into smaller modules arranged in a hierarchical structure, such that individual modules can be implemented on a single VLSI chip. The third objective of the thesis is to design a modular network where each module is responsible for recognition of only one particular sequence or subsequence. When a number of such modules are interconnected, the network can recognize longer sequences or detect a sequence of sequences.

1.5. Desired Input-Output Specifications

The neural network considered in this thesis has M inputs and one output, as shown in Fig. 1.1. In order to meet the main thesis objective, the network needs to have the following input-output characteristics.

- Input to the network consists of a vector of M continuous-time signals given by

$$Q = \{ q_1(t), q_2(t), q_3(t), \ldots, q_M(t) \} \qquad M \geq 1$$

where $q_i(t) \in R$, $i = 1,2,...,M$, and R is the set of real numbers.

- Input Q is sampled and held every T seconds, generating a sequence S of temporally related events. Let t_0 be the time of the first sample and E_i be the input event at $t_0 + (i-1)T$. The input sequence S can be represented by a vector given by

$$S = \{ E_1, E_2, E_3, \ldots, E_N \} \qquad N \geq 1$$

For the convenience of specification, the ith event of sequence S will be denoted as $E_i(S)$.

- Event E_i, $i = 1,2,...,N$ is represented by a vector of the form

$$E_i = \{ e_1, e_2, e_3, \ldots, e_M \} \qquad M \geq 1$$

where $e_j \in R$, $j = 1,2,...,M$ is the sampled and held value of input variable $q_j(t)$ at $t = t_0 + (i-1)T$. Input samples are assumed to be scaled such that $0 \leq e_j \leq 1$. The jth input

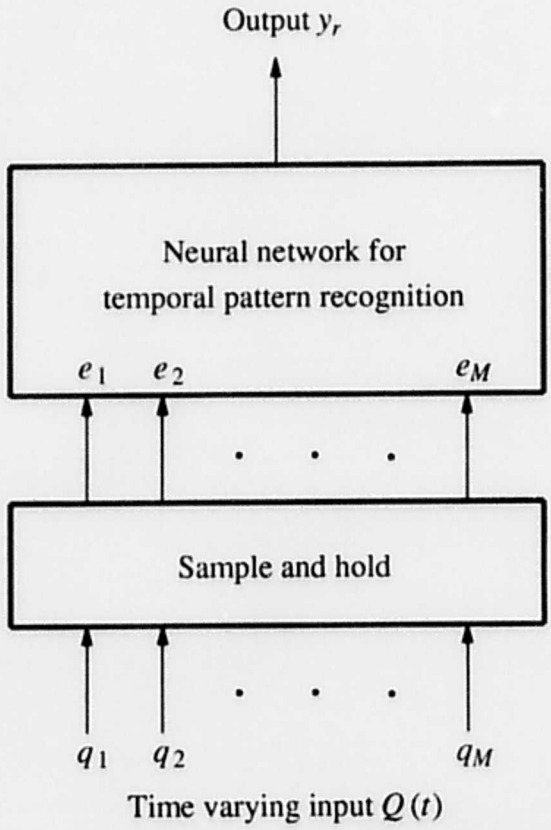


Fig. 1.1 A neural network with M inputs and one output for temporal pattern recognition.

variable of the ith event of sequence S will be denoted as $e_j(E_i(S))$.

- Let U_M denote the set of all possible input sequences for M inputs. A network is trained using a training sequence $l_s \in U_M$, and the performance of a trained network is evaluated using a test sequence $t_s \in U_M$. Sequence l_s and t_s are chosen such that

$$l_s \cap t_s \neq \varnothing$$

- We define $D(a, b) \in R$ to be the sum of the squares differences between corresponding events of two sequences a and $b \in U_M$ of equal length. That is

$$D(a, b) = \sum_{i=1}^{N} \sum_{j=1}^{M} (e_j(E_i(a)) - e_j(E_i(b)))^2$$

where $e_j(E_i(a))$ and $e_j(E_i(b))$ are sampled values of the jth input variable of the ith event of sequences a and b, respectively. Sequences a and b are regarded as the same if $D(a, b) = 0$. Sequence a is regarded as similar to b when $D(a, b) \leq \theta_1$, where $\theta_1 \in R, \theta_1 \geq 0$ is a threshold value. After a network has been trained to recognize sequence l_s, a test sequence t_s is regarded as a correct sequence when $D(l_s, t_s) \leq \theta_1$. It is regarded as an incorrect sequence otherwise.

- Let $a \in U_M$ and $b \in U_M$ be sequences of lengths n and N, respectively, and $n \leq N$. We define $P(a, b, k) \in R$ to be the partial difference between two sequences a and b as follows:

$$P(a, b, k) = \sum_{i=1}^{n} \sum_{j=1}^{M} (e_j(E_i(a)) - e_j(E_{i+k-1}(b)))^2$$

where $k \in I$, the set of integers, is an index such that $1 \leq k \leq N$ and $n + k - 1 \leq N$. Sequence a is regarded as a *partial sequence* of sequence b if $P(a, b, k) \leq \theta_2$ for some value of k, where $\theta_2 \in R, \theta_2 \geq 0$ is a threshold value.

- When a test sequence $t_s \in U_M$ is presented, the network is required to generate an output y_r representing the sequence, where $y_r \in R$ and $0 \leq y_r \leq 1$. The output remains 0 until enough samples are viewed, at which point the output starts to rise gradually from 0 to 1. The result of each event contributes only a small increment to the output, such that an output of 1 means that a full correct sequence has been presented. A partial sequence will cause the output to rise to a value between 0 and 1 to indicate that only parts of the correct sequence have been encountered. An incorrect sequence yields an output that is less than θ_3 where $\theta_3 \in R$, $0 \leq \theta_3 \leq 1$ is a threshold value.

  During a recognition process, incorrect events or input noise may be presented. On such an occasion, the output is desired to decay gradually to ensure that the network is robust in a noisy environment. By having an output that decays gradually under non-idealized conditions, a short burst of noise will not undo the effect of many successful samples that exhibit a trend. Thus, the network can classify sequences that have correct events separated by a few

incorrect events.

Let $c(E_i)$, $0 \leq c \leq 1$, represents the extent to which E_i has been recognized as a correct event and let x be a measure of the number of correct events presented. The desired behavior of the output y_r of the network can be expressed as follows:

$$\frac{\partial x}{\partial t} = \alpha c(E_i) - \beta x \qquad i = 0,1,2,...,N$$

$$y_r = f(x)$$

where α is a constant that determines the incremental rate of x when a correct event is presented, β is a constant that determines the decay rate of x when an incorrect event is received, and f is a nonlinear decision function.

1.6. Thesis Organization

The thesis is divided into 7 chapters. Chapter 2 presents background work on neural networks and existing approaches to temporal pattern recognition using neural networks. The proposed architecture is described in chapter 3, and the dynamics of node activation during temporal pattern recognition is analyzed in chapter 4. Chapter 5 describes the application of the network to the problem of motion detection in computer vision, together with simulation results. Serial and hierarchical interconnection of modules are discussed in chapter 6. A proposed implementation for the network on a silicon chip using current VLSI technology is presented in chapter 7. Conclusions and a summary of the contributions of the thesis are presented in chapter 8.

Chapter 2

Background

2.1. Introduction

This chapter discusses some of the background literature relevant to the topics of this thesis. Two main classes of neural networks and how they are trained are introduced, followed by some existing approaches to temporal pattern recognition using neural networks. The use of the temporal integration process in biological systems and in a high-level vision framework for motion detection is also briefly presented.

2.2. Neural Networks

Neural networks have recently become the focus of wide-spread research, not only due to their ability to perform computations in parallel, but also because of their ability to adapt to problems that might be otherwise very difficult to solve. They were originally proposed to model functions of the brain such as learning, association, classification, and concept formation based on anatomical and physiological characteristics of biological neurons in the brain. The first two models proposed were the McCulloch-Pitts logic element [McCulloch 1943] and the Perceptron [Minsky 1969]. The need for parallel processing of information in real time, which is difficult to achieve on a von Neumann computer, caused a change in the emphasis of research in this area [Rumelhart 1986, Mead 1989]. Current research on neural networks concentrates on designing adaptive systems for specific applications.

A neural network comprises a set of nodes, each of which has one output and a number of inputs [Kohonen 1984, Rumelhart 1986]. The inputs of one node are linked to the outputs of other

nodes by weighted connections, which may assume real values of either sign. Each node first receives and sums the activation or inhibition from other nodes via the weighted connections. Then, a nonlinear output in response to the sum is generated.

A neural network is trained to recognize a pattern by adjusting the weights of its interconnections. One way to program a neural network is by subjecting the network to a set of training patterns, and adjusting the weight of each connection in small increments based on certain learning rules [Hinton 1984, Ackley 1985, Rumelhart 1986, Rumelhart 1986b, Hinton 1989]. The weight adjustment is repeated until the network is able to generate correct responses when it receives the same training input patterns.

It was proven by Minsky and Papert that a simple network with only a single layer of nodes cannot be trained to recognize complex patterns [Minsky 1969]. For example, the training of a neural network to generate an output that is the exclusive-or of its two inputs is difficult because the state of one input does not provide any information about whether the output should be 0. It was shown later by Hinton that a network requires *hidden nodes* in order for it to be trained to recognize complex patterns [Hinton 1984, Rumelhart 1986]. The nodes in a network can be classified as either *visible* or *hidden*. Visible nodes are accessible to the environment and are divided into two disjoint classes: input nodes and output nodes. It is via these visible nodes that a network exchanges information with its environment. In contrast, the hidden nodes are not subjected to direct manipulation by the environment. They are affected only by their neighbors.

After a neural network has been trained, it starts to compute whenever an activity pattern is presented at its input terminals. During this period, the activity of each node is affected by the outputs of other nodes and the weights associated with the interconnections between the nodes. There are two classes of neural networks: *feed-forward* and *recurrent*. In a feed-forward neural network, node outputs are connected to other nodes downstream, with no feedback [Minsky 1969, Rumelhart 1986b]. In a recurrent neural network, a feedback arrangement is used, and the computation repeats for a few iterations until the network stabilizes to a final activity pattern containing the result [Hopfield 1982, Hinton 1984]. The following describes the architectures of these two

classes of neural networks.

2.2.1. Feed-Forward Neural Networks

A feed-forward neural network consists of one or more layers of nodes, where the outputs of the nodes in one layer are connected to node inputs of the next layer. For example, a two-layer network is shown in Fig. 2.1. The last layer consists of several output nodes and the intermediate layer consists of hidden nodes. There is no feedback between layers and the nodes in the same layer are not connected to each other. The nodes' outputs in one layer are determined by the inputs received from the previous layer and the weights of the interconnections linking the two layers. There is a clear forward direction of information flow from the inputs to the outputs of the network, and as a result, the network is inherently stable.

A feed-forward neural network is static because, except for a finite propagation delay, the outputs do not evolve or change once a pattern is presented to its inputs. The network's outputs are determined entirely by its current inputs. That is, there is no memory embedded in the network. Even though a feed-forward neural network lacks dynamics, it is quite powerful because it can be trained to respond to any input pattern through a learning procedure.

Analysis of a feed-forward neural network is manageable when there is only one layer of nodes, because the output of each node is influenced only by the external inputs and does not have any interaction from other nodes. However, the analysis becomes difficult when one or more layers of hidden nodes are added to the network. In this case, the output nodes receive nonlinear activation from hidden nodes in the previous layer, which in turn receive activation from another layer. That is, the output nodes are not influenced directly by the inputs.

Feed-forward neural networks have many applications [Fukushima 1980, Ballard 1984, Feldman 1985, Curlander 1987, Widrow 1988, Le Cun 1989]. For example, they have been successfully used in optical character recognition [Fukushima 1988, Le Cun 1989], where a textual bit-mapped image is transformed into ASCII codes by using a neural network trained to recognize the

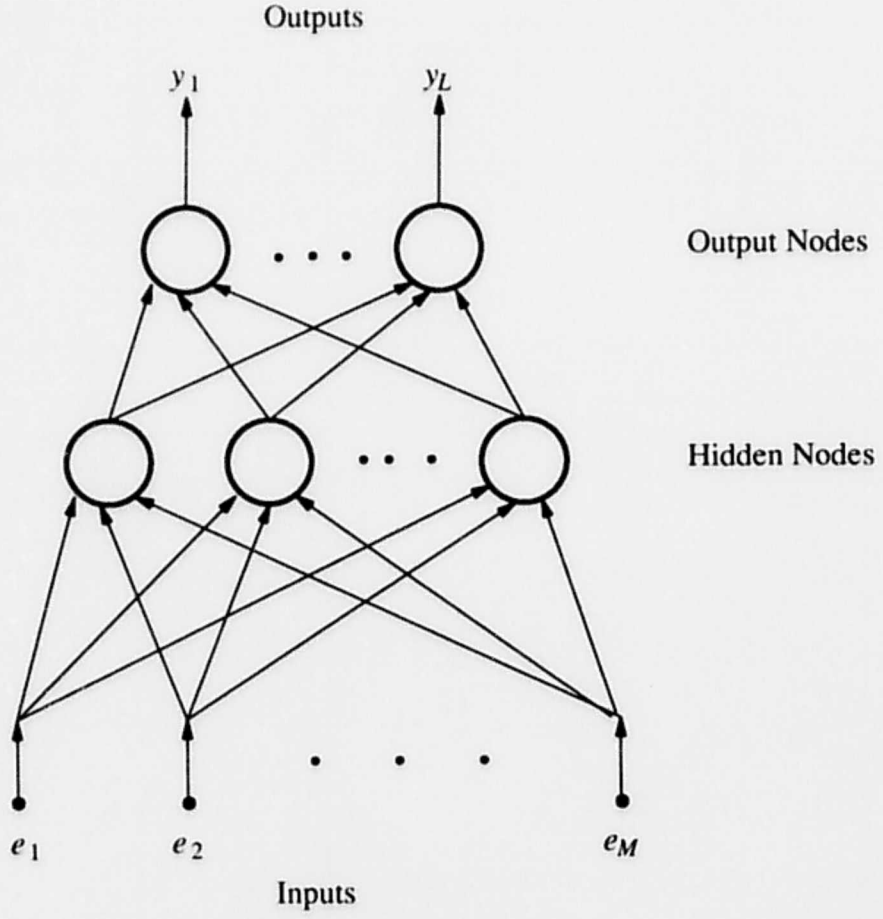


Fig. 2.1 A two-layer feed-forward neural network.

pattern of each ASCII character.

2.2.2. Recurrent Neural Networks

A recurrent neural network is a nonlinear dynamic system that contains feedback connections, as shown in Fig. 2.2. The input to each node is connected to the external input directly, as well as to the outputs of other nodes, by weighted summing junctions. Because of the feedback arrangement, the network outputs change with time.

Every node may have an internal delay used to compute the weighted sum of the outputs from other nodes and its external input. For example, in the Hopfield network, this internal delay is implemented by connecting a capacitor and a resistor at the input of each node [Hopfield 1982]. A recurrent network can also be implemented using nodes without any internal delay with memoryless nodes, in which case the network outputs are only computed in discrete time. An example of a recurrent network using memoryless nodes is the Boltzmann machine [Hinton 1984, Ackley 1985].

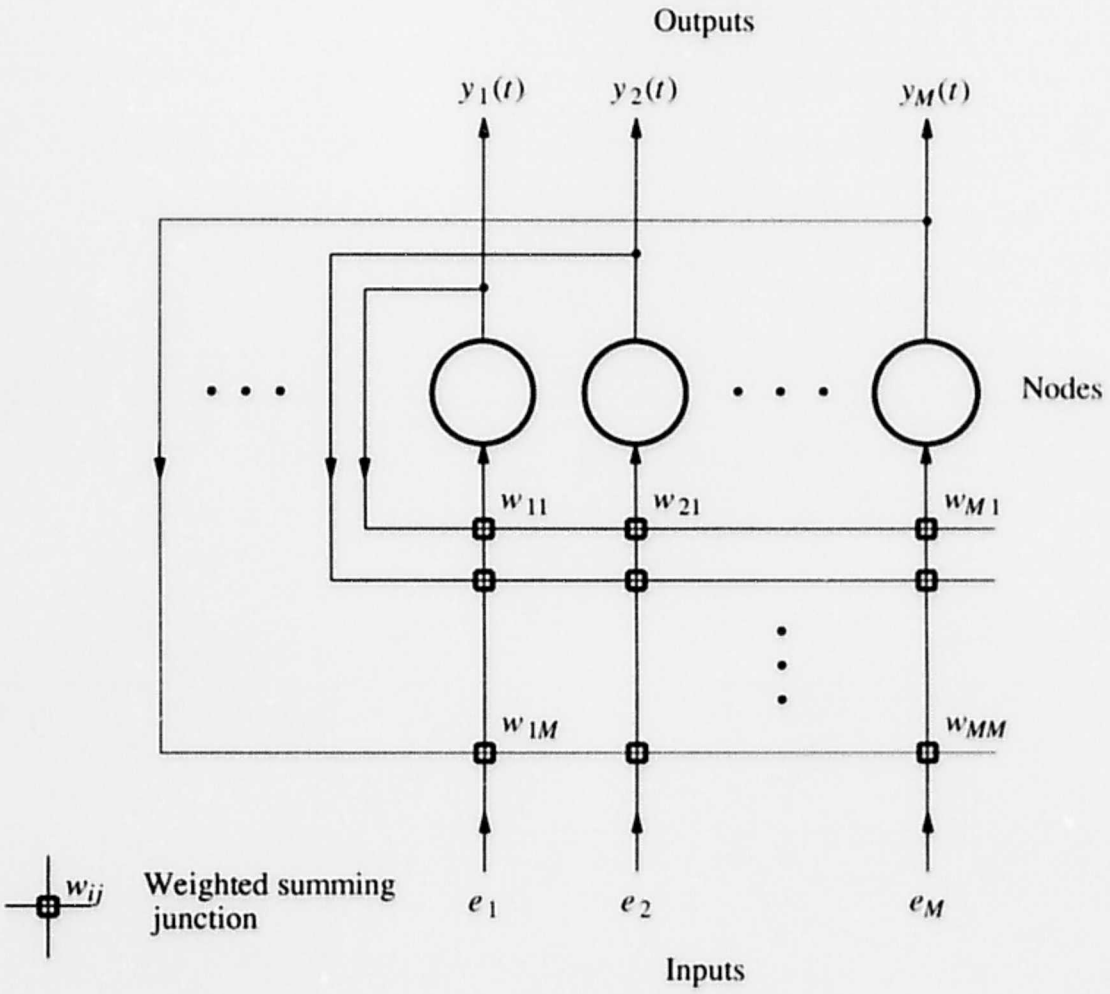


Fig. 2.2 A recurrent neural network with feedback connections.

The state evolution of a recurrent neural network can be very complex and sometimes chaotic. However, a recurrent network can be made stable and useful for information processing when the interconnections between nodes are symmetric (that is, each connection has the same weight in both directions) [Hopfield 1982]. This result was proven by Hopfield using the Lyapunov's Direct Method (reference can be found in [Slotine 1990, Wiggins 1990]). The state of a recurrent neural network with symmetric interconnections between nodes is always associated with a Lyapunov or energy function, V.

For example, the energy function of a given state of a binary recurrent network such as the Boltzmann machine is given by

$$V = -\sum_{i<j} w_{ij} y_i y_j + \sum_i \theta_i y_i \tag{2.1}$$

where w_{ij} is the weight of the connections between node i and j, y_i is 1 if node i is on and 0 otherwise, and θ_i is the bias associated with node i.

Starting from any initial state, a recurrent network will evolve in a direction that decreases the energy function, until the network reaches a stable state in which the energy function is at a local minimum. In general, a recurrent network has several stable states, sometimes referred to as fixed-point attractors.

For each attractor, there is an associated basin of attraction comprising a range of input patterns. Any input pattern that is within a basin of attraction will cause the network to evolve from a pre-defined initial state toward the corresponding attractor.

The mechanism underlying fixed-point attractors can be used to process information as follows. The network is first brought to the designated initial state. An input pattern is then presented to the network and is kept constant until the network reaches a stable state at one of the attractors. By decoding that state, an output can be used to produce the desired output corresponding to the input pattern.

The analysis of a recurrent network is not easy, due to the complex nonlinear interaction between nodes. The output of a node is influenced by both the inputs and the outputs of other nodes. This analysis is even more difficult when hidden nodes are included in a network.

Like feed-forward neural networks, recurrent neural networks also have many applications [Grossberg 1980, Hogg 1984, Hopfield 1988, McCelland 1986]. A popular one is a content-addressable memory. Patterns are stored inside a network as fixed-point attractors in a multi-dimensional state space. Once the patterns are stored, they can be recalled when an input pattern corresponding to one of the attractors is presented, even when there is noise in the pattern. The basin of attraction represents the extent to which a pattern can be distorted and still be recognized by the network.

2.3. Training Algorithms

A neural network can be trained to generate the desired output when its inputs are subjected to certain input patterns. A network is trained by an iterative procedure, in which successive training examples are presented and the interconnection weights are adjusted in small increments in the direction that minimizes a certain cost function. The cost function can be either the Lyapunov function described earlier [Hopfield 1982, Ackley 1985], or a measure of the difference between the actual and the desired outputs [Rumelhart 1986, Rumelhart 1986b, Widrow 1988]. The training phase is repeated until the cost function is at its minimum.

A number of training algorithms have been developed [Grossberg 1969, Fukushima 1975, Hopfield 1982, Hinton 1984, Kohonen 1984, Ackley 1985, Rumelhart 1986, Rumelhart 1986b, Hinton 1987, Hinton 1989, Williams 1989]. Each algorithm is well-suited to a specific network topology. Some training algorithms are described briefly below for the two main classes of neural networks presented earlier. These algorithms have been chosen because they are used to train the pattern classifiers in the neural networks for temporal pattern recognition, which will be presented in the next section.

2.3.1. Feed-Forward Neural Networks

Feed-forward neural networks may consist of one or more layers of nodes. The number of layers affects the type of training algorithm that can be used to train the network. First, we will examine how a single-layer feed-forward neural network can be trained using a popular algorithm known as the *Least-Mean-Square-Error* (LMS) algorithm, followed by the *Backward-Error-Propagation* algorithm, which can be used to train a multi-layer feed-forward network.

The LMS algorithm is generally a good choice for weight adjustment of a single-layer neural network when the input vector and the desired response of the network are available at each iteration [Tou 1974, Widrow 1985, Rumelhart 1986, Widrow 1988]. This algorithm is simple to implement because it does not require the computation of the gradient for the overall network function, as needed in other algorithms. The LMS algorithm will be utilized in chapter 3 to adjust the weights of the interconnections between nodes in the proposed network.

Consider a single layer of independent nodes each of which connected to the external inputs by weighted links, as shown in Fig. 2.3. The total input, x_j, to node j is given in the following

$$x_j = \sum_i w_{ji} e_i + \theta_j \tag{2.2}$$

where w_{ji} is the weight of the connection from the external input e_i to the input of node j, and θ_j is a bias term associated with node j.

The output of node j is given by

$$y_j = f(x_j) \tag{2.3}$$

where f is a nonlinear function that is continuous and differentiable.

The network is trained to respond to a set of pairs of input and output patterns. It uses the current weights and each input pattern to generate an output pattern. In each case, the actual output pattern is compared with the desired output pattern to determine the total error given by

$$H = \frac{1}{2} \sum_{i=1}^{P} \sum_{j=1}^{N} (\hat{y}_{ij} - y_{ij})^2 \tag{2.4}$$

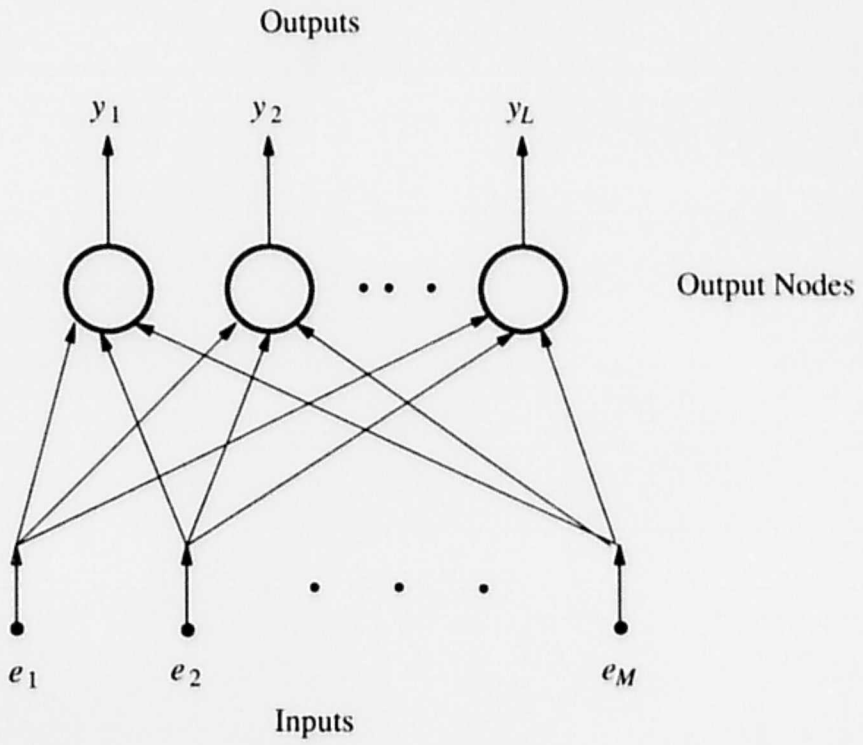


Fig. 2.3 A single-layer feed-forward neural network.

where $\hat{y}_{ij}$ and y_{ij} are the desired and actual outputs of node j, respectively, when the ith input vector is presented, N is the number of nodes in the network, and P is the number of input vectors in the training set.

The plot of the error function versus the weights is generally bowl-shaped, with one minimum point [Widrow 1985]. At each iteration of the LMS procedure the values of the weights are modified in small increments in the direction that reduces the error H, using the gradient descent method. The direction is given by the gradient of the error surface at the current point in weight space. Because of the bowl-shaped error function, the network can be trained easily and the LMS algorithm will find the weights that converge towards the point in weight space having the global minimum squared error [Widrow 1985].

The LMS algorithm works well at finding a set of weights that minimize the error function in single-layer feed-forward networks, but it is unable to determine interconnect weights in a multi-layer feed-forward network. The reason is that during the training phase, the desired output values of the nodes in the intermediate layers are not known, and their errors cannot be computed. The error function is no longer bowl-shaped, and the global minimum is difficult to find.

An extension to the LMS algorithm, called *backward-error-propagation*, was proposed to train a multi-layer network. To describe the algorithm, we define delta δ for a node to be the partial derivative of the error with respect to a change in the sum of all the weighted inputs to the node [Rumelhart 1986]. The delta for output node i is computed using the following rule

$$\delta_i = (\hat{y}_i - y_i) f'(net_i) \tag{2.5}$$

where $\hat{y}_i$ and y_i are the desired and actual node output, respectively, f' is the derivative of the non-linear output function, and net_i is the sum of the weighted inputs to node i.

The delta for hidden node j is computed recursively as follows:

$$\delta_j = f'(net_j) \sum_k \delta_k w_{kj} \tag{2.6}$$

where δ_k is the delta of node k in the layer above, and w_{kj} is the interconnect weight from node j to k.

The basic idea of the algorithm is to propagate back through the network deltas that are computed based on the actual and desired values of the output nodes. The deltas are first computed for the output nodes, and then propagated backward to all nodes pointing to the output nodes in the layer below. These nodes, in turn, propagate their received deltas backward to nodes pointing to them, and so on, until the input layer is reached. The interconnect weights are then changed according to the deltas, in much the same way as with the LMS algorithm.

There is a major drawback in the backward-error-propagation algorithm. Unlike a single-layer feed-forward network, the error function of a multi-layer network may contain many local minima. As a result, the gradient descent method may not find a set of weights that ensure a global

minimum of the squared errors. A feed-forward network may get stuck in local minima that are significantly worse than a global minimum, thus causing the network to perform poorly.

2.3.2. Recurrent Neural Networks

As in the case of feed-forward networks, a recurrent network can be trained by adjusting the interconnect weights between nodes to minimize an energy function, V [Hopfield 1982, Lapedes 1986, Almeida 1987, Rohwer 1987, Pinela 1988]. However, the energy function can be complex and may have many local minima. As a result, a recurrent neural network, in general, is difficult to train. A training algorithm that has been used to train a binary recurrent network called the Boltzmann machine is described briefly below.

The Boltzmann machine is a parallel computational network which consists of a set of binary nodes. The nodes are connected to each other via symmetric bidirectional connections. Conceptually, each node represents a simple hypothesis regarding a system about which the machine attempts to learn. If the node is on, the hypothesis is accepted. Otherwise, the hypothesis is rejected. The weights indicate the strength of the relation between pairs of hypotheses. Positive weights indicate that the hypotheses are correlated and negative weights mean that they are in some sort of conflict.

The network has a large number of states. The global energy function, V, associated with each state is given by Eqn. 2.1. This function has been chosen because it reflects the extent to which a given state of the machine violates the applied constraints. The problem is then to determine the values of the weights, w_{ij}, to minimize the energy of the system subject to the external constraints.

A simple way to minimize the energy is to toggle each node into one of its two states which yields the lower energy with all other nodes left unchanged. It has been shown by Hopfield that, if the nodes switch asynchronously and in the limit of negligible transmission times, the network will always settle into a local minimum of energy. The change in energy that occurs when the output of

node k switches from 0 to 1 or vice versa is given by

$$\Delta V_k = \sum_i w_{ki} y_i - \theta_k \tag{2.7}$$

In order to allow the network to escape from local minima, Hinton [Hinton 1984] adopted the Metropolis algorithm [Kirpatrick 1983] known as *simulated annealing*. Regardless of the present state of any node k, the node is switched on with a probability p_k given by

$$p_k = \frac{1}{1 + e^{-\Delta V_k / T}} \tag{2.8}$$

where T is a scaling parameter that acts like the temperature of a physical system. The idea is that if an occasional jump to high energy is allowed, one will avoid getting trapped in local minima. The network is first heated to a high temperature, that is, a large initial value of T is used, to allow a coarse search of configurations. Then, T is gradually reduced to allow the system to settle into a minimum.

The disadvantage of using simulated annealing is slow convergence towards a solution. It has been pointed out by Hinton that several thousand learning cycles are required to learn a simple problem [Hinton 1984].

2.4. Existing Approaches to Temporal Pattern Recognition

Input to a temporal pattern recognition system consists of a sequence of temporally related events, which have been generated by sampling a time-varying input at fixed time intervals. The system must have some memory elements that can briefly store information about individual input events in order to recognize the whole sequence. The neural networks presented in section 2.2, both feed-forward and recurrent, cannot, in general, be used for temporal pattern recognition. A feed-forward neural network is a static network, which can only process one pattern at a time. The result from the processing of one pattern has no influence on the processing of the next pattern. Similarly, the relaxation of a recurrent network to one of its attractors removes any dependence on initial conditions. The result only depends on the current input. Hence, the network is not suitable

for temporal pattern recognition, which requires that the output depend on both the current and previous inputs.

Two models have been proposed for temporal pattern recognition using neural networks. They differ in the way the information about a sequence of input patterns is stored. The first model stores the input information explicitly before it is processed by using a set of time delays, while the second stores the information as a state of the network. This section presents these two models.

2.4.1. Time-Delay Network

Both feed-forward and recurrent neural networks can be used as pattern classifiers. Because a classifier can handle only one pattern at a time, it is regarded as static, even though it may be internally dynamic when a recurrent network is used. A pattern classifier can be extended to recognize a sequence of events by providing a set of time delays to store the events in the sequence, as shown in Fig. 2.4 [Lang 1988, Waibel 1988, Widrow 1988, Bengio 1989, Lippmann 1989, Roitblat 1989, Tank 1989, Bottou 1990, Hampshire 1990]. The time delays are used to hold the K most recent events which together with the present input implement a fixed temporal window of size K. The delayed events combine to form a single static pattern, which is then processed by the classifier to generate an output representing the sequence.

The classifier's outputs are derived by computing the nonlinear weighted sums of the current and the deferred events, as follows.

$$y_l(kT) = \gamma_l \left(\sum_{i=0}^{K} \sum_{j=1}^{M} w_{ij} e_j(t-iT) \right) \qquad l = 1 \cdots L \tag{2.9}$$

where γ_l is the mapping function for output y_l of the pattern classifier, M is the number of inputs, w_{ij} is the weight of the interconnect from the jth input of the ith time-delay stage to the classifier, and $e_j(t-iT)$ is the jth input of an event that has been time delayed for iT period.

Time-delay neural networks have been widely used to recognize speech, and excellent performance has been obtained [Lang 1988, Lippmann 1989]. For example, in the work by Lang and

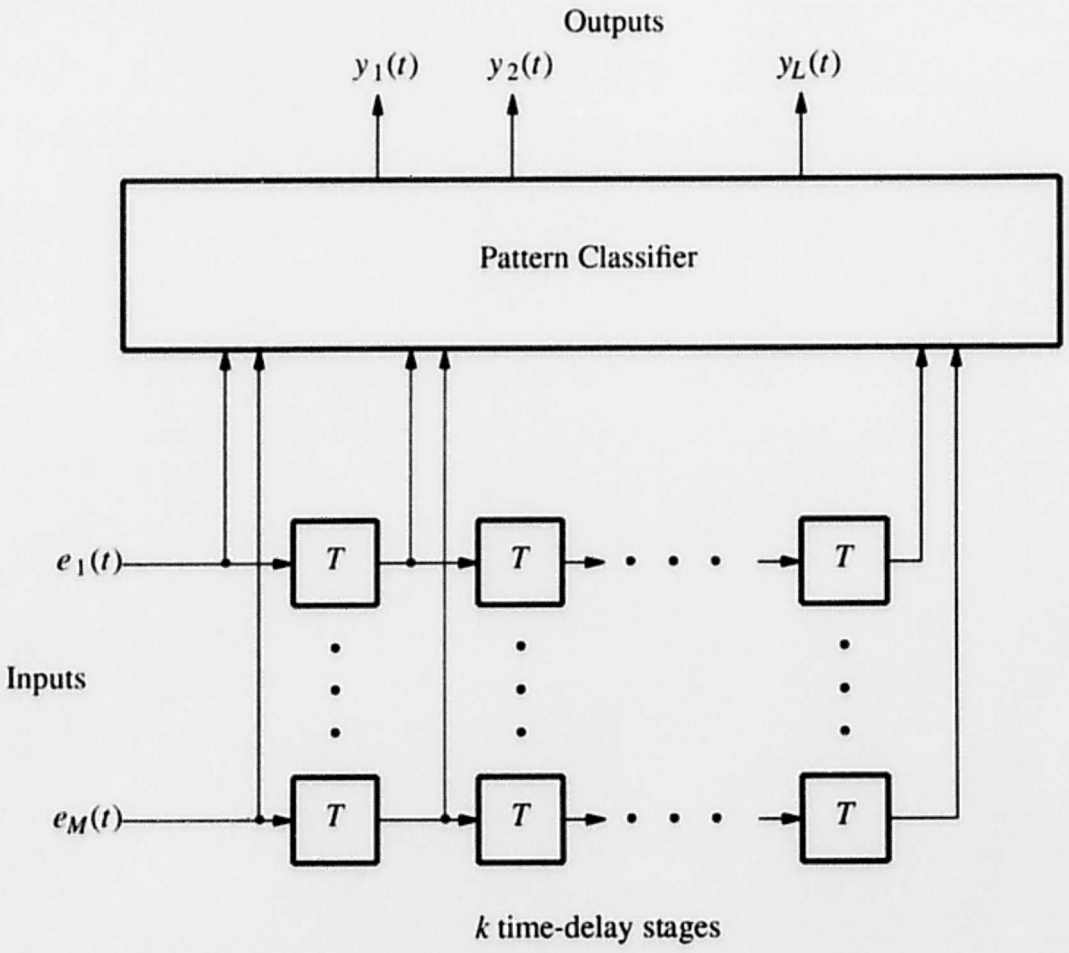


Fig. 2.4 A time-delay neural network model.

Hinton, a feed-forward network, together with a set of fixed-length time delays, is used to recognize four acoustically similar letters, *B*, *D*, *E* and *V* [Lang 1988]. The network is trained and tested using pre-segmented speech samples taken from the onset in these letters from 100 male speakers. The network is reported to achieve an accuracy of 91.4 % when speech samples taken from other speakers are presented.

Tank and Hopfield obtained satisfactory results when a recurrent neural network, together with a set of time delays, was used to recognize words in a continuous stream of speech [Tank 1987]. In one of their simulations, a network of 50 nodes is trained to recognize the names of 50 of

the states in the United States of America. When a random sequence with an embedded sequence such as *IDDEHO* is presented, the node representing the state *IDAHO* is activated.

2.4.2. Sequential Neural Network

A sequential network is an abstract model consisting of a number of states, a set of input vectors, a set of output vectors, a next-state function, and an output function, where input, output and state values are defined only for integer values of time. The network's current state and the current input, together, determine the state of the network at the next period. When there is a finite number of input vectors, output vectors and states, the network is termed a finite-state machine, which is a common component in a digital system.

The general model of a sequential network for temporal pattern recognition is given in Fig. 2.5. It comprises a pattern classifier and a feedback network. Some of the classifier outputs are fed back to the network inputs through time delays. Therefore, the next state and the outputs of the network in Fig. 2.5 can be determined by the current state $F(t) = f_1(t),...,f_P(t)$ and the current inputs $E(t) = e_1(t),...,e_M(t)$, as follows

$$F(t+T) = \psi(F(t),E(t)) \tag{2.10}$$

$$Y(t+T) = \gamma(F(t),E(t)) \tag{2.11}$$

where ψ and γ are the next-state and the output functions of the pattern classifier, respectively.

The pattern classifier may be a feed-forward network or may be a recurrent network. In the latter case, the classifier itself would also contain internal time delays and feedback. However, as explained earlier, the internal time delays in the classifier do not store information from one input pattern to the next. For the purpose of this discussion, the outputs of the classifier will be assumed to settle in a period shorter than T. It is important to note that a sequential network is a recurrent network, because of the feedback connections. However, unlike the classifier operation, the input vector is allowed to change before the overall network reaches steady state. Sequential networks used for temporal pattern recognition have been referred to simply as recurrent networks in some

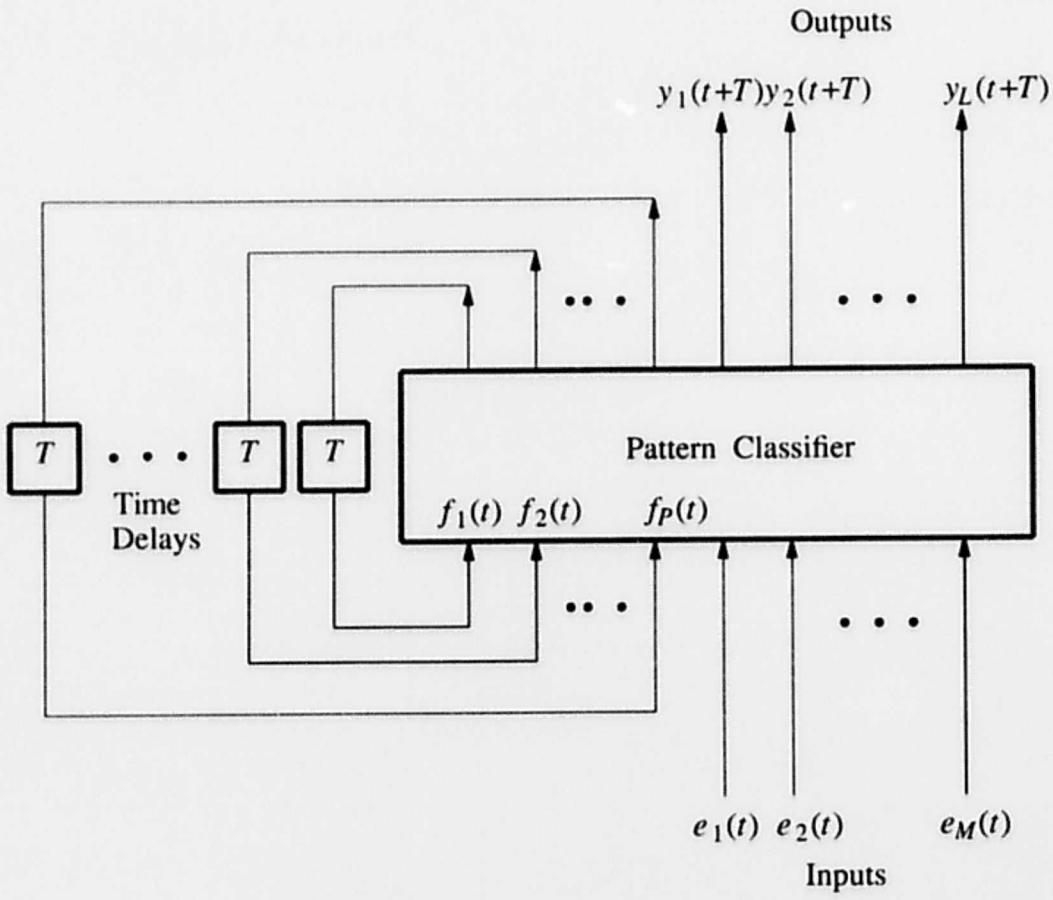


Fig. 2.5 General model of a sequential network.

literature [Lippmann 1989]. However, to distinguish between the pattern classifier and temporal pattern recognition uses of a recurrent network, only the network in Fig. 2.5 will be called a sequential network in this thesis.

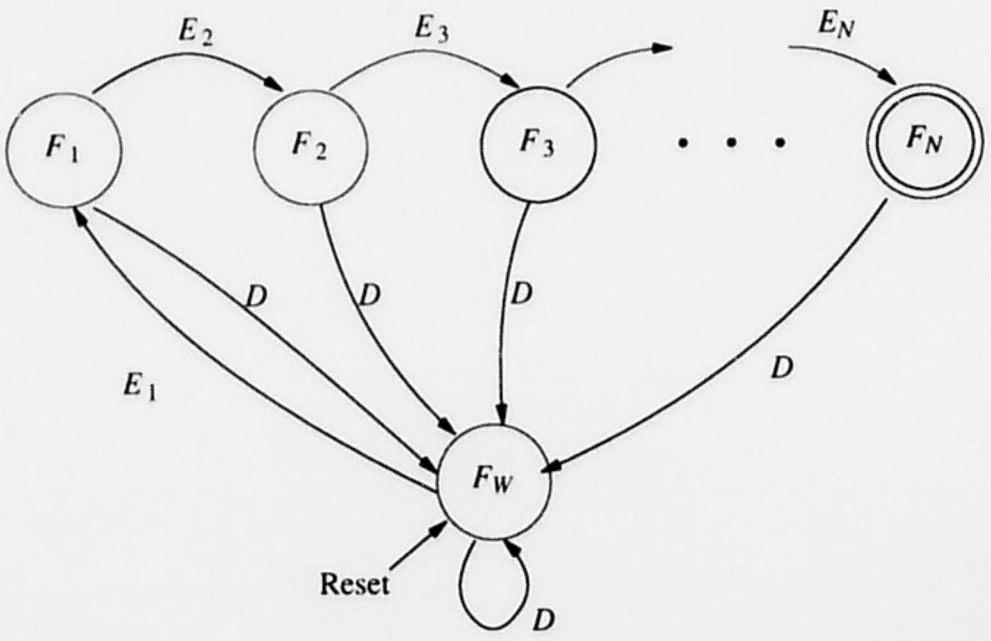
A sequential network possesses many states, which can be used to recognize an input sequence. The evolution from one state to another can be used to represent the arrival of one input pattern after another in a sequence. Thus, a sequence of input events can be associated with a certain sequence of network states. Suppose that a sequential network begins with a designated initial state and is supplied with an input sequence. If, after the last event is presented, the network reaches a state that generates a certain predesignated output, then the network is said to have

recognized that input sequence.

The behavior of a sequential network can be described by the state transition diagram, shown in Fig. 2.6. The label of each directed arc specifies the input event that causes the network to change from one state to another. The network always waits in its idle or wait state F_W for the first event, E_1, to arrive. After event E_1 is presented, the network exits its idle state and enters state F_1. The presentation of E_2 at the next period will cause the network to leave state F_1 and arrive at state F_2. As a result, a correct sequence causes the network to evolve from the idle state F_W to F_1, F_2, etc. State F_N is the accepting state that represents the recognition of a sequence and is indicated by two circles. Any input noise will bring the network to the idle state F_W, from which the network will have to wait for the arrival of event E_1 to begin another recognition process. State F_N is not reached until N events in the sequence have been presented in the correct order, spaced T seconds apart.

Prager, Harrison and Fallside implemented a sequential network similar to the one shown in Fig. 2.5. to recognize vowels [Prager 1986]. They used a Boltzmann machine (a binary recurrent neural network) to implement the classifier shown in the figure. In their system, analog input data in the form of a speech spectrum are converted into binary representations and presented to the network. A number of output nodes are used to specify the vowels. When a sequence of speech spectrum of a vowel is presented to the network, the network evolves from one state to another until a final state is reached in which only one of the output nodes is excited. After training using speech samples taken from 6 speakers, the network achieved a success rate of 85 %.

Jordan has used the recurrently-connected multi-layer feed-forward network structure shown in Fig. 2.7 for guiding robotic motion and for speech generation [Jordan 1986]. The network uses a layer of self-feedback hidden nodes called context nodes to record the temporal information of previous events in a sequence. These nodes may have lateral interconnections between them (not shown). The outputs of the context nodes, together with the external inputs, feed into the inputs of a multi-layer feed-forward network, whose outputs are in turn fed into the context nodes. Jennings and Keele used the network proposed by Jordan to recognize a temporal sequence of numerical



F_W : wait state

D : disturbance due to incorrect event or noise

Fig. 2.6 State transition diagram of a sequential network to recognize a sequence of events.

digits [Jennings 1989]. The focus of their work is the ability of the network to distinguish between sequences that contain unique digits in all positions, such as {1 5 4 2 3}, and sequences that contain repeated digits such as {3 1 2 1 3 2}. Elman used a similar network to explore the representation of structure in time [Elman 1990]. A multi-layer feed-forward network that has recurrent links in its hidden and output nodes was used by Watrous to capture spectral/temporal characteristics of phoenetic features [Watrous 1987].

Stornetta, Hogg and Huberman used a layer of recurrent input nodes, each of which has a weighted self-feedback loop, to implement a sequential network [Stornetta 1987], as shown in Fig. 2.8. The feedback allows previous inputs, together with the current input, to influence the output of the node. Since analogue signals are used, each node behaves as an infinite impulse response

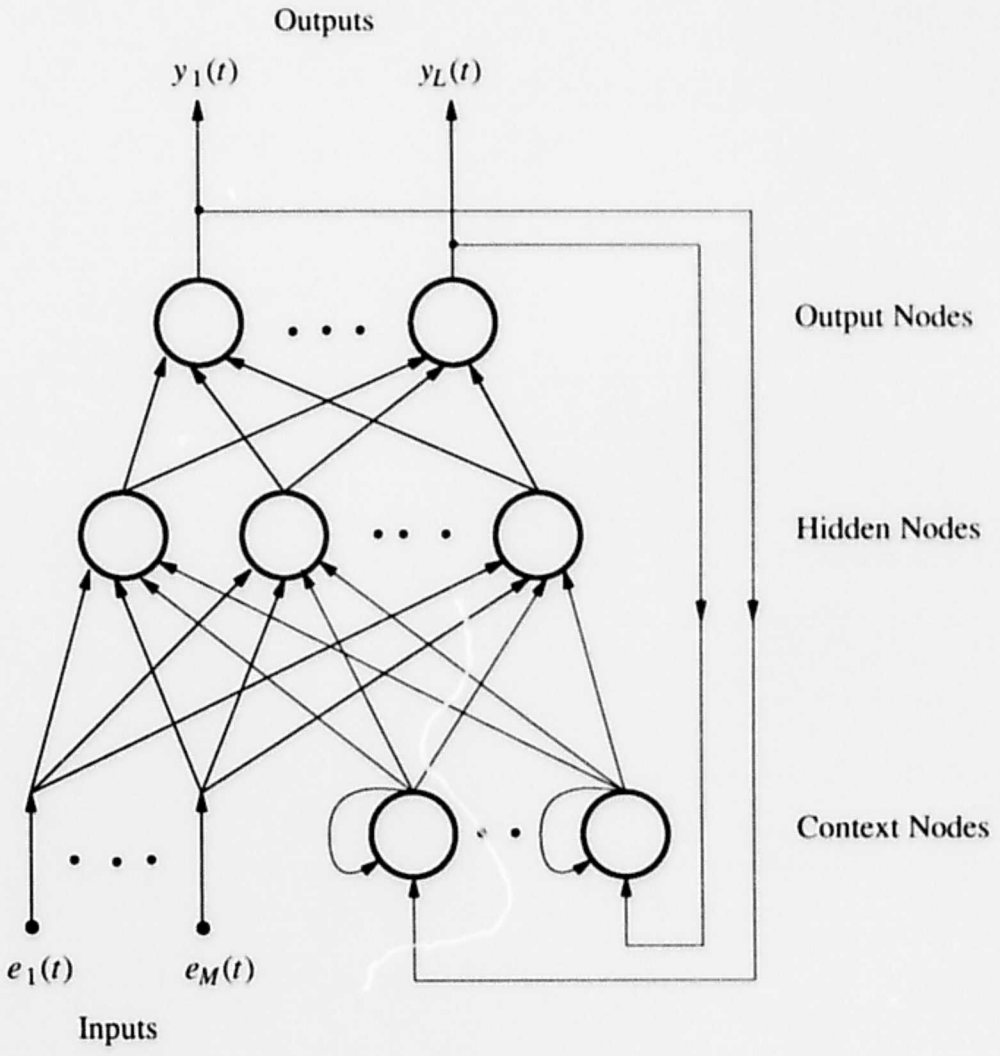


Fig. 2.7 A recurrently-connected multi-layer feed-forward network.

digital filter [Bose 1985, Rabiner 1975]. A time-dependent pattern is observed at the outputs of the input nodes when a sequence of patterns is presented. A feed-forward neural network is used to classify the outputs of the nodes at fixed time intervals in order to recognize a sequence. This network has been applied to one-dimensional motion detection by training the network to detect leftward or rightward motion of a gaussian pulse moving across the field of input nodes.

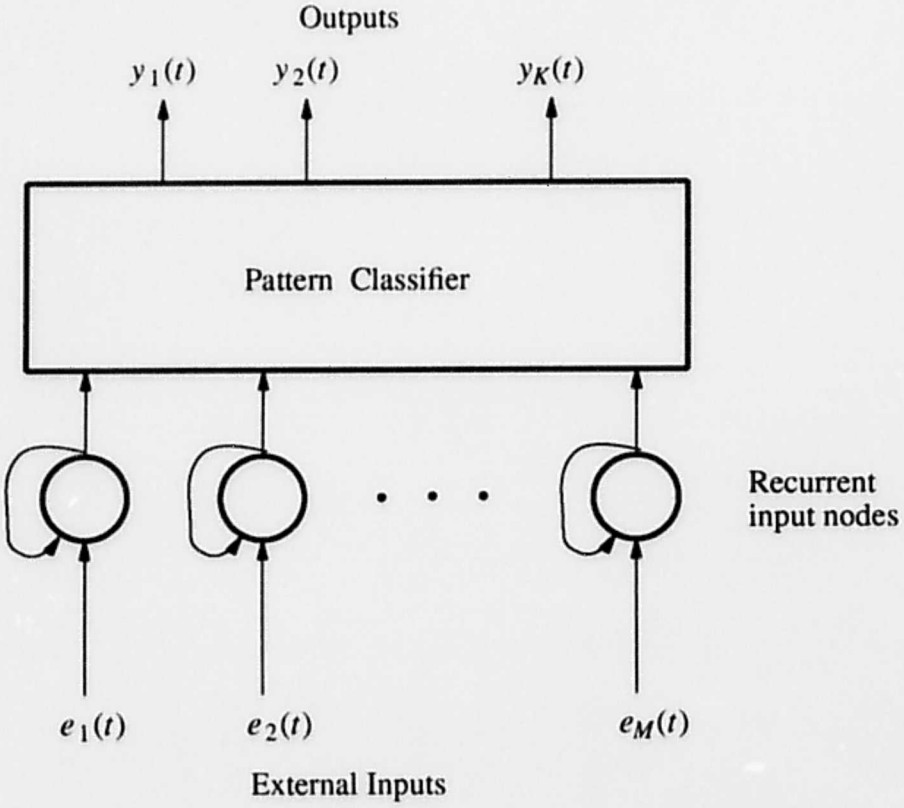


Fig. 2.8 A sequential network implemented by a set of input nodes with self feedback.

The sequential network proposed by Shamma uses a similar layer of input nodes to implement a bank of filters, to model the frequency responses of the biological filters in the cochlea [Shamma 1989]. Unlike Stornetta, Shamma uses a winner-take-all pattern classifier to detect the edges and peaks in the spatial pattern defined by the outputs of the recurrent input nodes. His classifier uses a layer of laterally inhibited nodes.

2.5. Temporal Integration Process

This section describes a computational process that can be used to overcome some of the limitations of the two existing models for temporal pattern recognition. This process, which is known

as temporal integration, is commonly observed in biological systems [Kandel 1985]. It was utilized by Tsotsos in a high-level computer vision framework for motion understanding [Tsotsos 1980, Tsotsos 1987].

2.5.1. Biological Systems

The ability of a brain to process time-varying inputs provides us with the reason to examine its basic information processing mechanism closely. Although the functioning of a brain is still a mystery, a great deal is known about the properties of individual neurons.

The brain is an organ composed largely of nerve cells or neurons. A neuron is connected to another neuron by a connection called synapse. The signals received by one neuron from other neurons are tempered chemically depending on the strength and type of the synapses. The signals are summed and thresholded by a neuron to generate an output.

The cell membrane in each neuron has passive resistive and capacitive properties, which have important effects on the flow of electrical signals within a neuron. These passive electrical properties contribute to the time constant of the neuronal membrane, which affects both the rise time and decay time of the internal activity of the neuron. That is, a neuron changes from a quiescent state to an active state, or vice versa, according to a time constant. This property provides the neuron with a fading memory, which enables a neuron to account for the passage of time.

Because of the neuron's time constant, its state of excitation is determined by the time integral of the sum of its inputs. The process of computing this time-integral is called *temporal integration*. A long time constant means that signals which are far apart in time can interact. The effect of the time constant is illustrated in Fig. 2.9. A presynaptic cell A is driving two postsynaptic cells B and C, where the time constant of cell C is ten times smaller than cell B. Fig. 2.10 gives the responses of the internal activities of both cells. Due to its large membrane time constant, cell B has a higher chance than cell C of being triggered by cell A. That is, incoming signals that in isolation may be too weak to trigger a neuron can sum in time to reach the required thres-

hold, provided that the distance between them is shorter than the time constant of the cell.

In summary, the time constant and the connections between neurons give neurons the ability to sum their input signals both spatially and temporally. The spatial and temporal behavior of a collection of massively connected neurons enables the brain to process complex time-varying inputs. This observation motivated the use of temporal integration in neural networks to recognize a sequence of events.

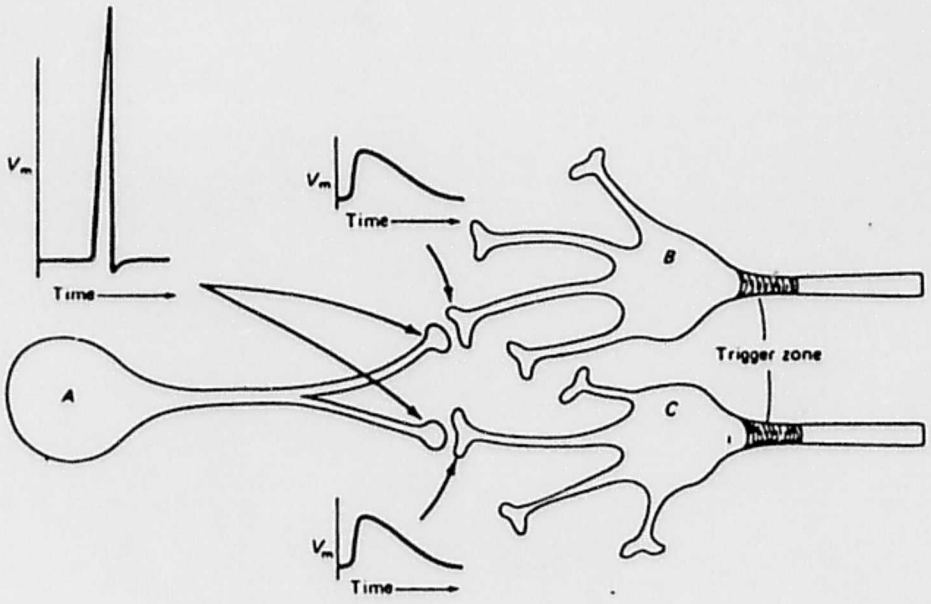
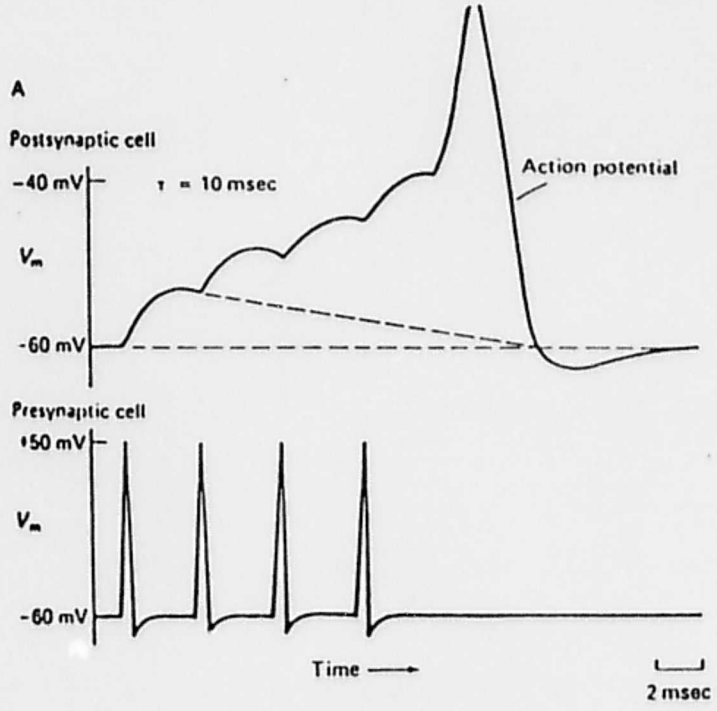
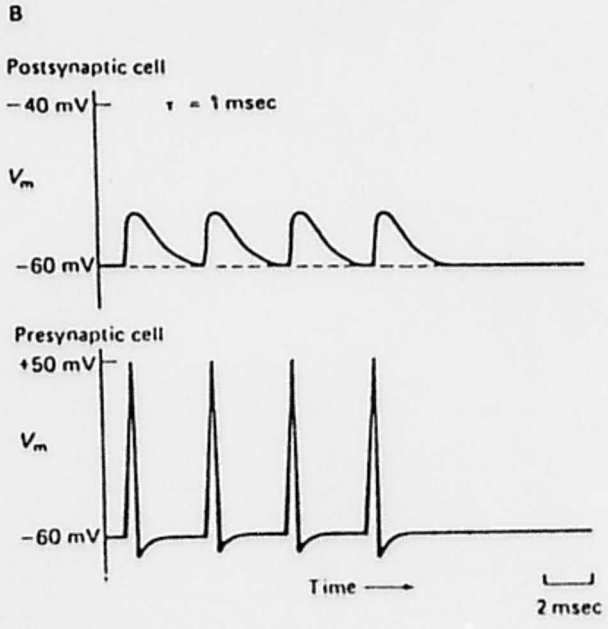


Fig. 2.9 A presynaptic cell driving two postsynaptic cells with different time constants (taken from Kandel 1985). V_m is the membrane potential.



a) Response of cell *B*



b) Response of cell *C*

Fig. 2.10 Responses of postsynaptic cells with different time constants when they are driven by a presynaptic cell (taken from Kandel 1985).

2.5.2. High-Level Vision Framework

In computer vision, a framework that incorporates time into a model for high-level vision can be found in ALVEN — an expert vision system that evaluates the performance of the human left ventricle from a sequence of X-ray images [Tsotsos 1980, Tsotsos 1984, Tsotsos 1987]. Conceptual description of the shapes and motions exhibited by the left ventricular wall is abstracted from a sequence of X-ray images and presented to ALVEN for detection of unusual occurrences in a patient's heart.

Abnormalities of the left ventricle are detected by comparing the motion of the left ventricular wall of a damaged heart with that of a normal heart. The presentation of the first image in a sequence causes the activation of a set of hypotheses about the occurrence of subsequent images in the sequence. The image hypotheses are temporally related to each other by the *NEXT* and *PREVIOUS* links, as shown in Fig. 2.11. They are also related to another hypothesis that represents the overall sequence by a *PART–OF* link. Each of these links are weighted connections with a real value of either sign. The weight indicates the strength of correlation between the two hypotheses.

Once activated, the image hypotheses are compared with the sequence of images to check for features that support the hypotheses. Certainty of the current hypothesis about an image is calculated based on whether the image matches the expected features of the hypothesis and whether the previous hypothesis supports the current hypothesis. The certainty value is close to 1 when both conditions are true. It is close to 0 otherwise. This process of certainty computation is performed for every image in a sequence using a relaxation scheme.

The certainty of every hypothesis about the corresponding image in a sequence is weighed by the *PART–OF* link and then temporally integrated to compute the certainty of recognizing the whole sequence.

The ALVEN framework addresses several requirements that are crucial to temporal pattern recognition, and hence is relevant to this thesis. In particular, the passing of time is represented explicitly by the sequence of hypotheses H_1, H_2, ... etc and the *NEXT* and *PREVIOUS* links

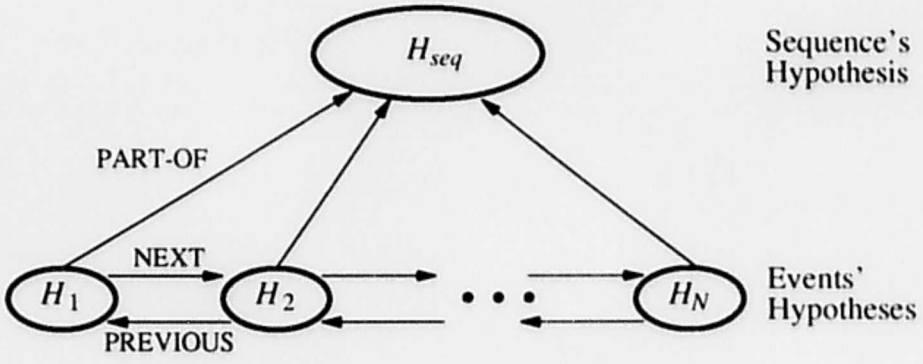


Fig. 2.11 Hypotheses generation used in ALVEN.

between them. Also, temporal integration of the *PART–OF* information is used to compute the final result. Thus, the final output is based on the sum over time of the certainty of each hypothesis. The use of temporal integration to generate the final output gives ALVEN robustness in a noisy environment and also the ability to detect partial sequences.

Although ALVEN is not a neural network in the current sense, the interpretation of its structure provided the inspiration for the network organization proposed in this thesis.

2.6. Concluding Remarks

Both time-delay and general sequential network models have been widely used in many applications. Unfortunately, at present, they have limitations when they are required to meet the input-output specifications given in chapter 1. The reasons and problems encountered will be discussed in the following chapter. This thesis proposes a special configuration of a sequential neural network that utilizes the process of temporal integration to overcome the problems. The proposed network will be presented in chapter 3, along with a comparison of the network with some existing networks.

Chapter 3

Proposed Approach

3.1. Introduction

This chapter proposes the use of a special configuration of a sequential neural network to recognize a sequence of temporally related events. The process of temporal integration is utilized in the network to generate a response representing the sequence. The purpose and intended difference of the proposed approach from existing approaches is five-fold. The network is: modular, easy to train, tolerant to noise, is able to recognize partial sequences, and is easily analyzed. The proposed network uses a special case of a sequential network with one memory element per state to satisfy the above requirements.

This chapter begins by presenting some limitations of existing neural network models for temporal pattern recognition, followed by the proposal of a neural network model that utilizes a temporal integration process to recognize a sequence. The algorithm used to train the proposed network to recognize a sequence, and how a number of modules can be interconnected and trained to recognize multiple sequences are presented. Some simulation results that demonstrate the functioning of a temporal module and a comparison of the proposed network with other existing neural networks for temporal pattern recognition are also given.

3.2. Limitations of Existing Network Models

The main objective of this thesis is to design a neural network which generates an output that rises in small increments from 0 to 1 when a correct sequence is presented, and decays in small decrements to 0 otherwise. This output behavior provides the network with the ability to recognize

partial sequences and to operate in a noisy environment.

Two neural network models have been proposed for temporal pattern recognition: the time-delay and sequential networks. Both models have shown great promise in such applications as speech recognition. They may be adapted to meet the above requirements, but the question is how well and at what cost? While they do have a structure that could be tailored, at the moment neither network is able to meet the desired input-output requirements.

In a time-delayed neural network, the events in a sequence are deferred in time to form a static pattern, which is then processed by a pattern classifier. Hence, it can only recognize a sequence after the entire sequence has been presented. As soon as the first event of the next sequence is presented, the output will be immediately reset. There is no gradual rise and decay of the network output.

In a sequential neural network, state transitions are used to represent the arrival of the events in a sequence. Starting from an idle state, if the network reaches an accepting state after all events have been presented, then the sequence is recognized. The need to present all events means that the network output remains reset until the entire sequence has been presented. As soon as the accepting state is reached, the output is set to 1 indicating recognition. The output is immediately reset when the next sequence is presented. Again, there is no gradual rise and decay of the output.

A gradual system response is preferred to an all-or-none response because it gives the system better noise tolerance and also the ability to recognize partial sequences. A system with an all-or-none response cannot tolerate any missing event or input noise in a sequence, whereas a system with a gradual response is able to generate a partial response when only parts of the sequence are presented or when there is noise in the input.

In a biological system, the temporal integration process seems to play an important role in the ability of the brain to process time-varying input. Likewise, the high-level vision framework proposed by Tsotsos makes use of the temporal integration process to generate a gradual response representing a sequence. This process gives the system the ability to recognize partial sequences,

and makes it robust in the presence of noise.

In principle, both the time-delay and sequential neural networks can be tailored to include the use of a temporal integration process to recognize a sequence. This can be achieved by training the pattern classifier to generate an output that constantly reports whether the current event presented is correct. The results from recognition of individual events in a sequence can then be integrated temporally, to generate a time integral that represents the correct events received. The integral is then applied to a nonlinear decision function such that the final output remains 0 until enough samples are viewed before the output starts to rise from 0 to 1.

An implementation of a time-delay neural network with temporal integration is shown in Fig. 3.1. It has a pattern classifier with N outputs, where N is the number of events in a sequence. The outputs of the classifier are in turn connected to the inputs of an integrator R. Output y_i is set to 1 when correct events $E_1, E_2, ..., E_i$ have been presented to the network. Otherwise it is reset to 0. Therefore, a correct sequence will cause the outputs to be activated one at a time, from y_1 to y_N. A response representing the sequence can be generated by temporally integrating these output activities of the pattern classifier. An incorrect sequence causes all outputs to remain quiescent, and the time integral of these inactive outputs gives rise to a zero response.

An implementation of a general sequential neural network with temporal integration requires a pattern classifier that generates N outputs, as shown Fig. 3.2. Output y_i is set to 1 when events E_{i-1} and E_i are correct, and is reset to 0 otherwise. When a correct sequence is presented, the outputs will be activated from y_1 to y_N, one at a time. As a result, the final response representing the sequence can be generated by computing the temporal integral of the pattern classifier's outputs.

There is a major problem with these customizations. Both neural network models have an already unwieldy pattern classifier. It would be even more so if it were modified to provide an output that indicates the recognition result of each event in a sequence. This requirement increases the number of constraints that the network must satisfy, which means that a larger number of hidden nodes would be needed. The error or the energy function of the resulting pattern classifier can have

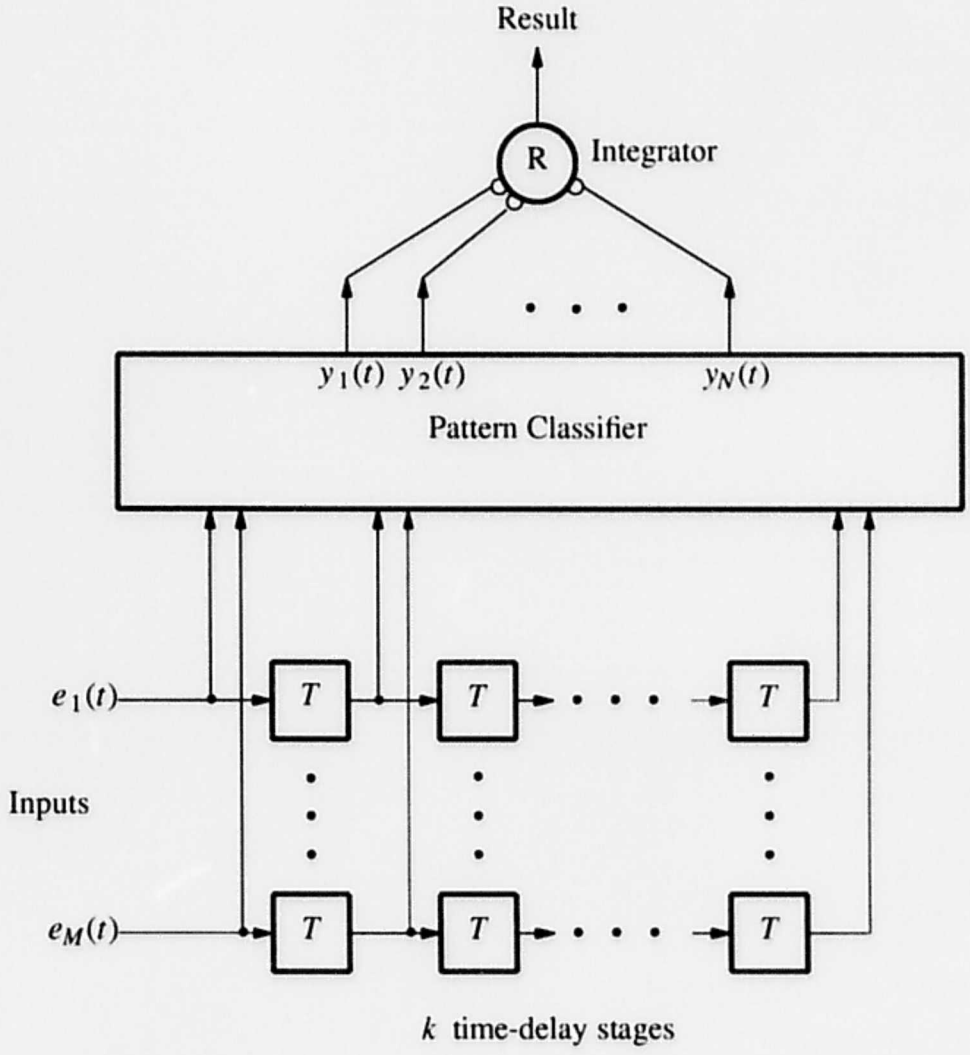


Fig. 3.1 A time-delay neural network with its outputs connected to an integrator R to compute the time integral of the result of each event.

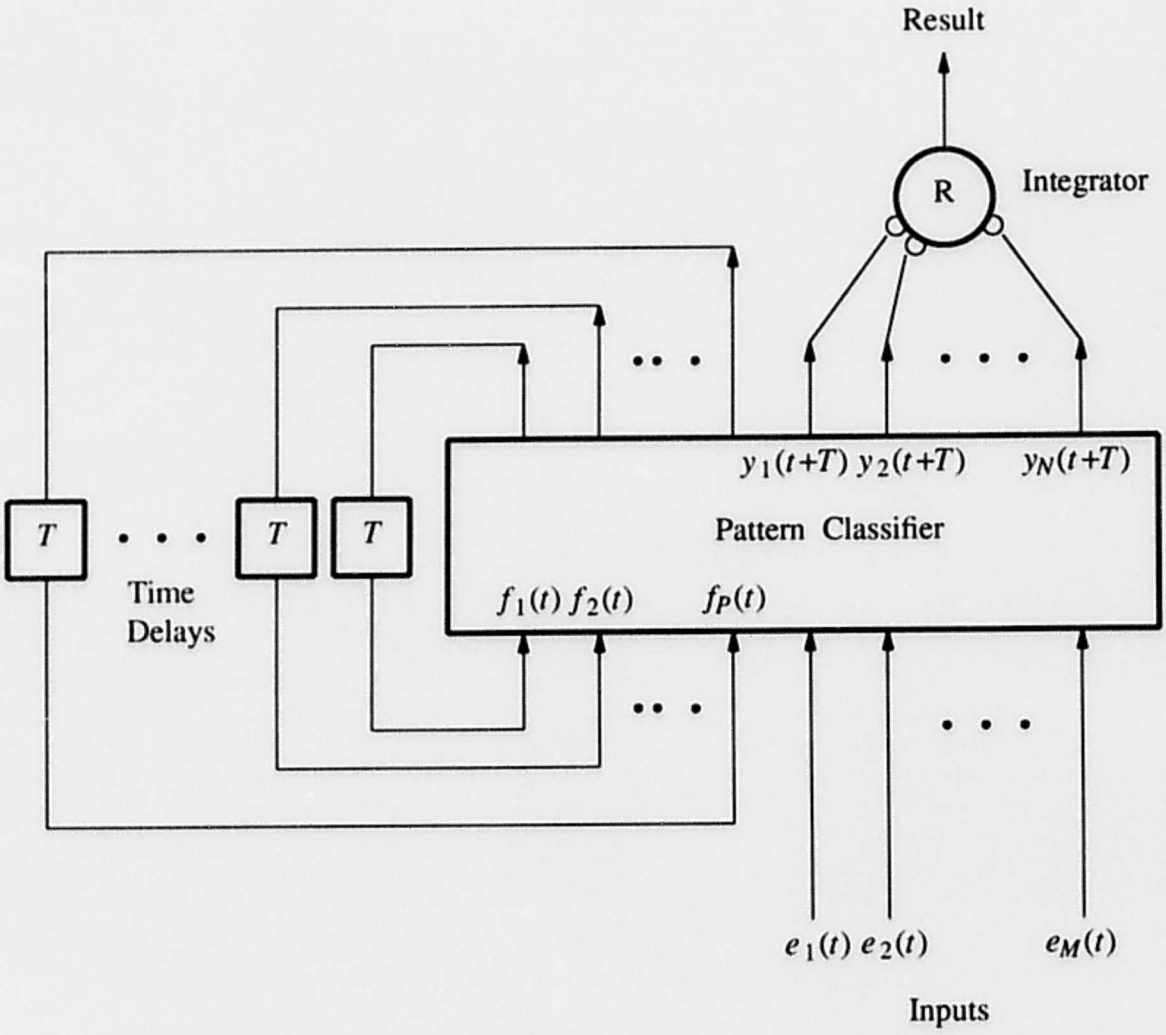


Fig. 3.2 A general sequential neural network with its outputs connected to an integrator R to compute the time integral of the result of each event.

many local minima which would make training a difficult process. A neural network with many hidden nodes is also difficult to analyze. At this time, there is still no underlying mathematical theory to analyze such networks.

As will be presented in section 3.7, another difficulty with the time-delay network model is the inherent scaling problem; a large number of time delays are required when the number of inputs per event and the number of events in a sequence are large. In such a case, the pattern

classifier would have many weighted interconnects to adjust during a training session, and require a longer time to train.

The general sequential network model does not have the scaling problem. However, it has several other problems. A general sequential network has many states that are not entered when a correct input sequence is presented. An incorrect input due to noise, wrong events, or imperfect learning, can cause the network to enter one of these states. From there, the network may follow an incorrect trajectory. As a result, the network is intolerant to non-idealized conditions. Another point is that a correct sequence may be embedded in other sequences and may start at any time. These other sequences, prior to the appearance of the correct sequence at the network inputs, cause the network to follow an incorrect trajectory. When the correct sequence is presented, the network may start from one of the incorrect states and follow an entirely different trajectory, causing recognition failure. Therefore, in order to introduce noise tolerance and the ability to recognize a sequence that may start at any time, the incorrect states need to be recognized to reset the network to its designated initial state. This increases the number of constraints that the classifier must satisfy, thus increasing the need for hidden nodes and the complexity of the training process.

3.3. Proposed Network Model

Our objective in this section is to develop a special configuration of the sequential network that overcomes the limitations of the general sequential network model discussed above. This section begins by presenting the architecture of the proposed network, followed by a description of the design of a node.

3.3.1. Architecture

The pattern classifier of the general sequential neural network in Fig. 3.2 is required to generate the state and the output of the network at the next period based on the network's current state and the current input. This requirement increases the number of constraints that the classifier must satisfy, which means that a complex classifier with several hidden nodes is needed. One way to

simplify the pattern classifier is to separate the functions of input event recognition and next state generation. That is, a pattern classifier can be used to recognize the input events and a state machine to account for the temporal order of the recognized events.

The structure of the enhanced sequential network is shown in Fig. 3.3. The network consists of a pattern classifier, a state machine, and a temporal integrator R. The state machine comprises a number of nodes S_1, S_2, ... and S_N. Unlike the general sequential network, this network has no feedback through the pattern classifier which is used strictly for recognizing the pattern in an input event. Furthermore, the sequential machine has a special structure that uses one memory element per state. The function and operation of nodes S_1, S_2, ..., and S_N will be described first, followed by node R. Nodes S_1, S_2, ..., S_N will be named the *Sequence* nodes, and R the *Result* node.

Output o_i of the pattern classifier is connected to the input of sequence node S_i by an excitatory connection. There are also excitatory connections from sequence nodes S_1 to S_2, S_2 to S_3, etc., forming a forward chain. The state of excitation of each node changes with a fixed time constant, which provides the node with a fading memory that introduces the time element used to process a temporal sequence. A chain of interconnected sequence nodes is responsible for recognizing the temporal order of events in a sequence.

Each output of the classifier is trained to recognize only one event of the sequence. When event E_i is presented, output o_i associated with the event is activated, and causes node S_i to be partially activated. Node S_i in turn sends a positive excitation signal to the next node on the chain, S_{i+1}, via the excitatory connection, which causes node S_{i+1} to be partially activated. if the next event of the sequence is that associated with classifier output o_{i+1}, then node S_{i+1} will be activated further. Otherwise, the output of node S_{i+1} decays to zero within a certain time constant. Node S_i has an inhibitory input from node S_{i+1}. Thus, when node S_{i+1} is excited, node S_i is turned off. As a result, when a correct sequence is presented, the nodes are activated one at a time, from left to right. The time constant of the fading memory must be larger than the sampling period of the inputs to allow the results of the recognition of successive events to be summed (see Section 2.5.1 in Chapter 2).

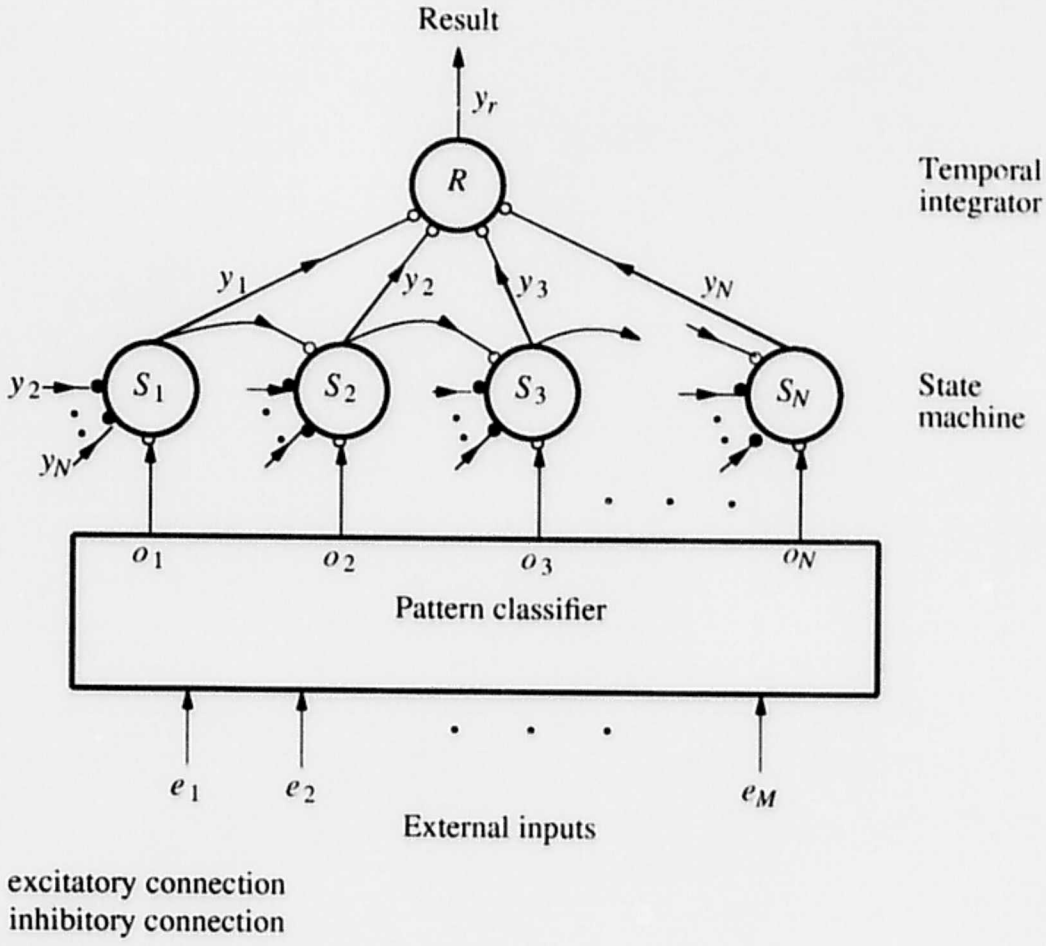


Fig. 3.3 The proposed network.

The use of a chain of interconnected nodes in the proposed network model to record the temporal order of events in a sequence was inspired by the work of Barlow and Levick [Barlow 1964, Barlow 1965, Dowling 1987]. They proposed a chain of interconnected nodes to account for the motion-sensitive property of ganglion cells in rabbits.

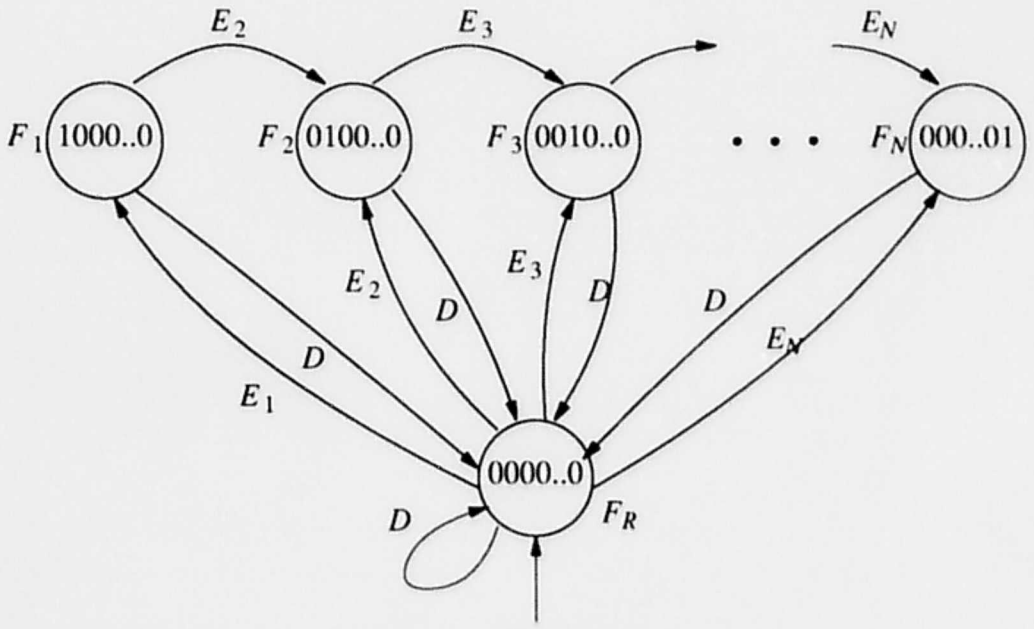
No more than one node can be active at any given time as a correct sequence is presented. However, incorrect states, that is states entered as a result of non-idealized input conditions, have more than one active node. These incorrect states can cause the network to follow an incorrect trajectory and lead to erroneous recognition. Hence, they need to be recognized and inhibited. This is

accomplished by operating the nodes in a winner-take-all fashion, so that there can be only one node active at any time. This is achieved by connecting each node to all other nodes, except its upstream neighbor, by inhibitory connections. As a result, either a correct event leads to exactly one node being activated or all nodes are inhibited bringing the machine to its reset state. Therefore, the proposed network in Fig. 3.3 is clearly a special case of a general sequential network. It uses a separate memory element (i.e. a separate node) for each state along the state trajectory that corresponds to a correct sequence.

In an ideal case, the state transition diagram of the network is as shown in Fig. 3.4. The network always waits in reset state F_R for the arrival of a correct event. After event E_i is presented, the network exits the reset state and enters state F_i, in which the ith node is set and the other nodes remain reset. If the next event presented is E_{i+1}, then the network will evolve from state F_i to state F_{i+1}. However, if an incorrect event is presented, the network will return to the reset state. As a result, a full correct sequence causes the network to evolve from reset state F_R to F_1, F_2, F_3..., and finally F_N. In the case of a partial sequence, the network will evolve from reset state F_R to F_i, F_{i+1}, ..., and F_j, where $1 \leq i < j \leq N$. An incorrect sequence causes the network to remain in state F_R.

Because the network has recurrent connections, the network's operation also can be described in the same manner as a nonlinear dynamical system. That is, the network has a fixed-point attractor near the origin of a N-dimensional phase space. When a correct sequence is presented, the network state leaves the attractor, evolves along a designated trajectory associated with the sequence, and then returns to the attractor after the presentation of the last event. The trajectory is approximated by a vector sequence $\{F_i, F_{i+1}, \ldots F_j\}$, when events E_i, E_{i+1}, ... and E_j are presented to the network, one event at a time, where $1 \leq i \leq j \leq N$. However, an incorrect sequence would cause the network state to evolve along a trajectory different from the designated trajectory.

Since a correct sequence will cause the nodes to be activated from left to right, one node at a time, and an incorrect sequence will cause most nodes to remain reset, the time integral of the weighted sum of all node outputs provides the required information about the degree of confidence in recognizing a given input sequence. The computation is performed by connecting the node



E_i : ith event

D : disturbance due to incorrect event or noise

Fig. 3.4 State transition diagram of the proposed sequential network.

outputs to the inputs of the result node R by weighted excitatory connections, as shown in Fig. 3.3. The weighted outputs are summed and integrated with respect to time in node R. A complete correct sequence causes the internal state of the integrator to rise from 0 to 1. For a partial sequence, the internal state of the result node will rise to some value between 0 and 1.

The result node also has a nonlinear output response to its internal state. It becomes active, that is, its output rises above 0.5, when the internal state exceeds 0.5 . The functional parameters of the node determine the number of correct input events that need to be recognized before the result node is activated.

The temporal integration process must be designed to reject correct events separated by many incorrect events. This may be achieved by using a "leaky" integrator. That is, if activation of node S_i, with all other nodes inactive, represents the correct recognition of input event E_i, and if x_r is the

internal state of the integrator, the desired behavior can be expressed in the form:

$$\tau_r \frac{dx_r}{dt} = \sum_{i=1}^{N} c_i y_i - x_r \tag{3.1}$$

where τ_r is the time constant of the result node, c_i is the weight of the interconnection from the output of node S_i to the result node. The rate at which previous correct events are forgotten is $\frac{1}{\tau_R}$. If each event contributes equally to the final response, then weight c_i is the same for all interconnections.

Because each output is activated only for one input event, the pattern classifier in Fig. 3.3 can be replaced by N independent classifiers, each attached to one sequence node, as shown in Fig. 3.5. The simplified classifier at each sequence node is implemented by a set of weighted connections, which may be either excitatory or inhibitory, from the external inputs, $e_1, e_2, ..., e_M$ to the node.

A null input event may be embedded in a sequence that a network is trained to recognize. In the network of Fig. 3.5, this is handled by using an extra input q, which is connected to the output of an external null input detector and has a weight of d_j. Input q is set to 1 when the event presented to the network is null, and 0 otherwise. A sequence node can be trained to recognize a null input event by adjusting the weight d_j of the connection from input q to the input of node j.

Consider a network of N sequence nodes interconnected as shown in Fig. 3.5. The nodes all have the same time constant, τ_s. Taking into account the total weighted inputs to a node from other nodes and external inputs, the dynamics of the internal state of each sequence node, x_j, can be expressed in the form

$$\tau_s \frac{dx_j}{dt} + x_j = \sum_{k=1, k\neq j}^{N} a_{jk} y_k + \sum_{k=1}^{M} b_{jk} e_k + d_j q + \theta_j \tag{3.2}$$

where a_{jk} is the weight of the connection from the output of sequence node k to the input of sequence node j, b_{jk} is the weight of the connection from the external input e_k, q is the input terminal connected to the output of the null input detector, d_j is the weight of the connection from the null input terminal to sequence node j, and θ_j is the bias of node j.

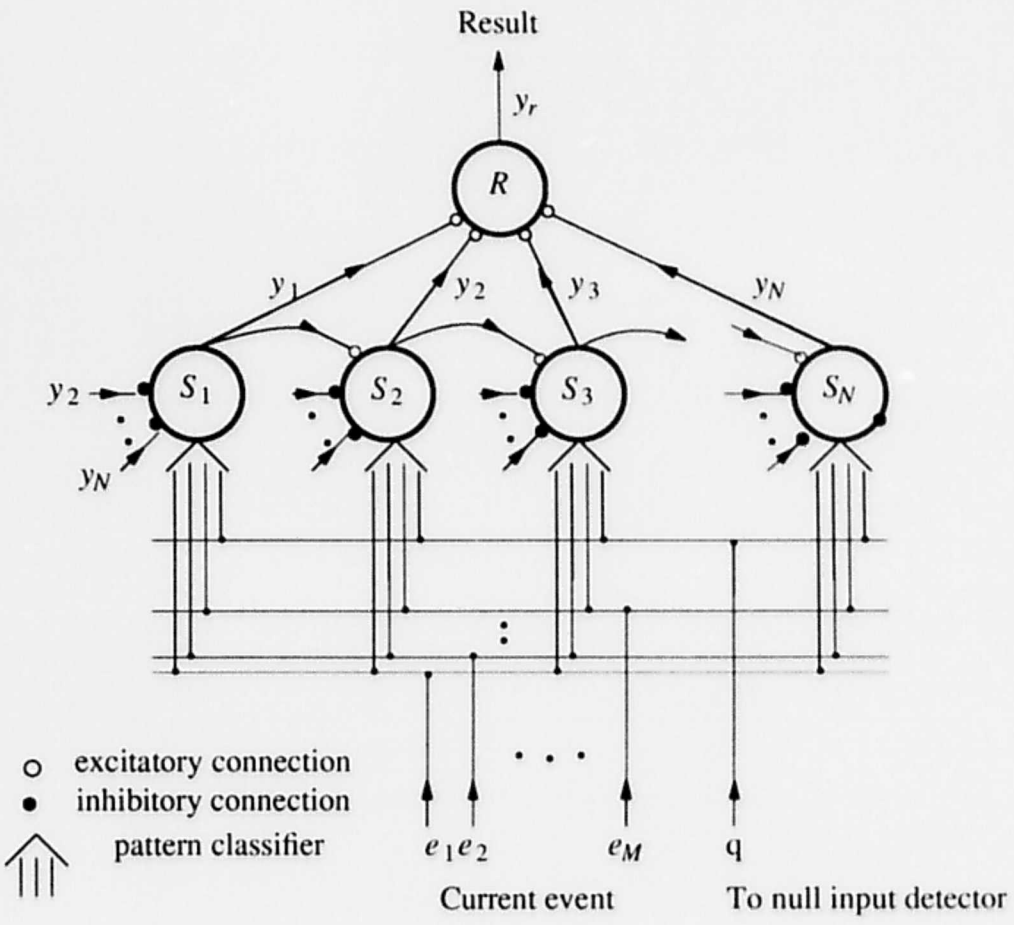


Fig. 3.5 Simplified view of the proposed network.

The use of one memory element per state and a simple pattern classifier at each sequence node makes it possible to analyze the state transition dynamics of the network, as will be discussed in Chapter 4. The analysis yields acceptable ranges for machine parameters, thus providing a systematic approach to design.

3.3.2. Design of a Node

A node has one output and several inputs. Each input may be connected to the output of another node or to an external input by a weighted connection. It has an internal state that changes with a fixed time constant and provides the node with a fading memory. The internal state, x_j, of

node j satisfies the equation

$$\tau \frac{dx_j}{dt} + x_j = \sum_k^N a_{jk} y_k + \sum_k^M b_{jk} e_k + \theta_j \tag{3.3}$$

where τ and θ_j are the time constant and internal bias of the node, respectively. The output, y_k, of node k is connected to the input of node j, by weight a_{jk}. External input e_k is connected to the input of node j by weight b_{jk}. Node j is connected to N nodes and M external input lines. The weight of each connection is modifiable and may assume values of either sign. A connection with a positive weight is referred to as an excitatory connection and a connection with a negative weight is referred to as an inhibitory connection.

Each node has a sigmoidal output in response to its internal state,

$$y_j = \frac{1}{1 + e^{-x_j}} \tag{3.4}$$

which is shown in Fig. 3.6. The sigmoidal function is used because it is a non-linear function that exhibits the threshold behavior needed for decision making. It is preferred to a hard limiter function because its derivatives, which are used in the training algorithm, are continuous and easily computed. The function also provides a close approximation to the transfer function of an amplifier, as will be explained in Chapter 7.

The output of each node has a continuous value between 0 and 1 and a median of 0.5 at $x_j = 0$, as shown in Fig. 3.6. In this thesis, a node is regarded as active whenever its output is higher than the level of 0.5 and inactive otherwise.

3.4. Training of the Proposed Network

The use of one memory element per state allows all the intermediate states of the network in Fig. 3.5 to be enumerated during a training session. As a result, the network can be trained to recognize a given sequence of events by adjusting the interconnection weights using the LMS algorithm described in Appendix A. The target values of the state variables were given in Fig. 3.4.

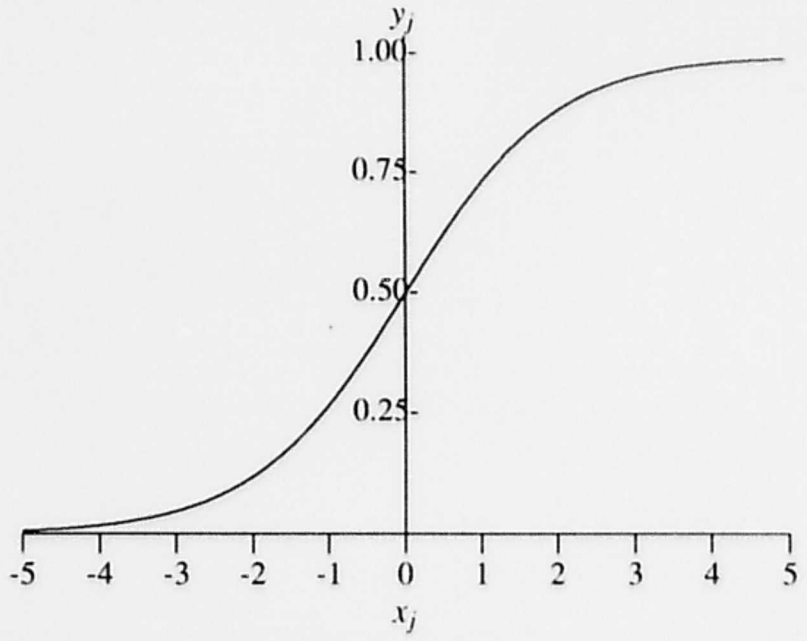


Fig. 3.6 A sigmoidal transfer function

Even though the sequence nodes are mutually interconnected, a simple LMS algorithm can still be used, because the nodes are arranged in a single layer and the desired node outputs are known at all times. For the same reason, the LMS algorithm can be used to train the result node. Note that the LMS algorithm presented in the following is an ad hoc training scheme. Modifications are made in order to impose a certain structural organization in the proposed network.

This section presents the two phases of the training procedure for the sequence and result nodes in the proposed network.

3.4.1. Sequence Nodes

Several parameters of the sequence nodes in Fig. 3.5 need to be adjusted so that the nodes are able to keep track of the events in a sequence. They are: the weights of the connections from the

external inputs to all the sequence nodes; the weight of the connection from the output of one sequence node to another; and the bias of each node. The time constant for each node is fixed and is defined by $T \leq \tau_s \leq 2T$.

Before we examine how the weight of each of these different connections is adjusted, the general rule for weight adjustment will be considered first. For each event in a sequence, the weights should be adjusted such that only the corresponding sequence node becomes active. That is, output y_i becomes equal to 1 when event E_i is presented. In each step of the iterative training process, the weights are updated based on the errors in the response compared to the desired values. Upon presentation of an event, the weight w_{ij} of the connection from any source j to the input of node i is updated using the following rule

$$w_{ij}(t+1) = w_{ij}(t) + \eta y_i (\hat{y}_i - y_i)(1 - y_i) e_j \tag{3.5}$$

where η is a small constant, $\hat{y}_i$, y_i are the desired and actual outputs of node i, respectively, and e_j is the external input, which is assumed to be constant during each sampling period. Refer to Appendix *A* for the derivation of this rule.

While training the network for event i, which should activate sequence node i, node $i-1$ is assumed to remain active from event $i-1$, due to the memory incorporated in the sequence nodes. At the same time, the activation of node i should cause node $i+1$ to be partially activated, due to the excitatory connections from node i to node $i+1$. All other nodes are assumed to be inactive. Thus, the desired responses of the sequence nodes as successive events of a training sequence are presented as given in Fig. 3.7. In the first period, the first two nodes are partially activated, and all the other nodes remain reset. In the second period, the second node is fully activated and the third node is partially active, and so on until the $N-1$st period, at the end of which node N is fully activated.

The weights of the connections of the links between the external inputs and sequence nodes are adjusted using Eqn. 3.5 at each time step. The bias of each node can be regarded as a permanently active source with an adjustable weighted connection to the node. This weight is adjusted

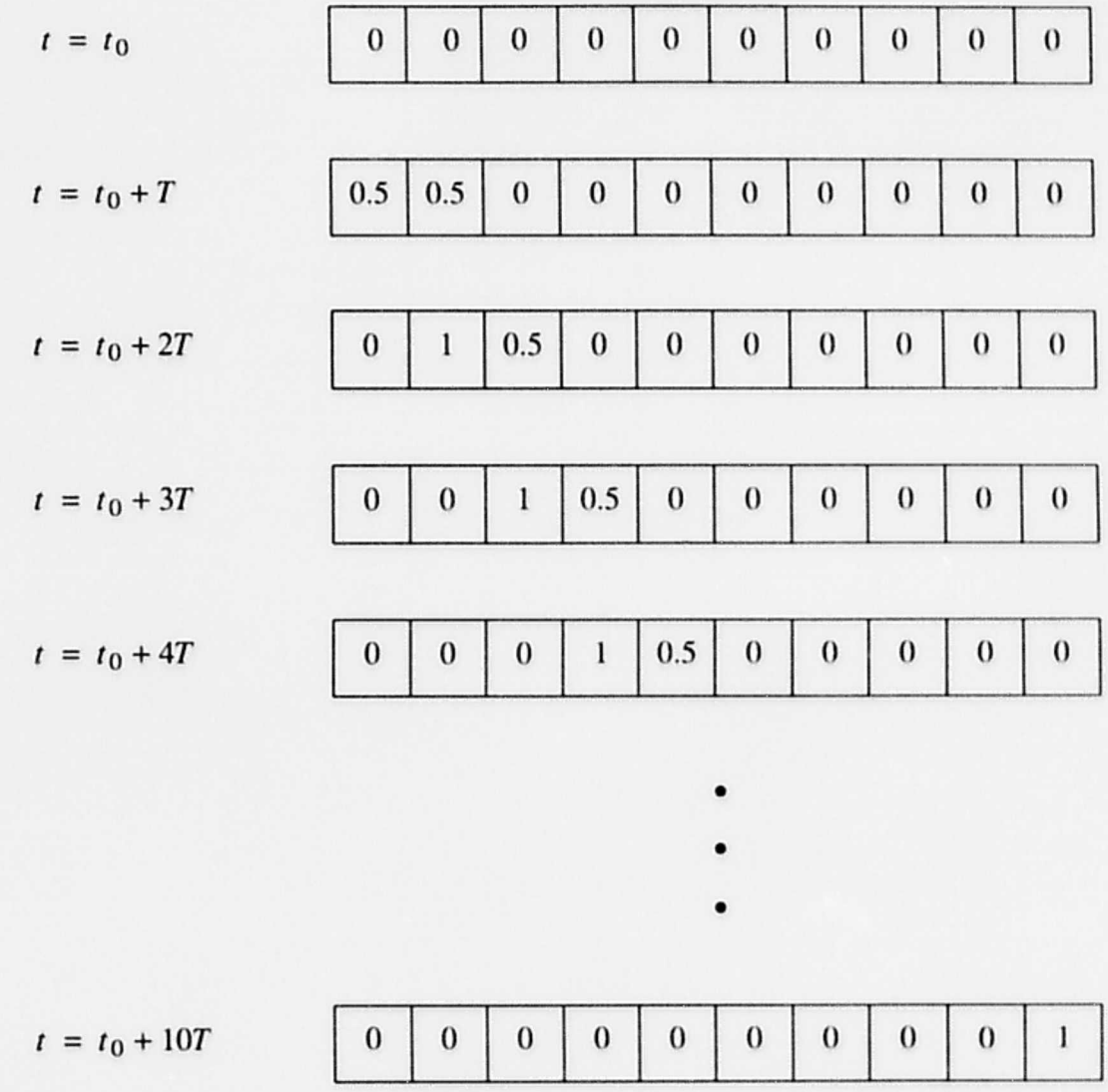


Fig. 3.7 Desired response during training.

in the same way as the weights of other connections to the node.

The weight of the excitatory connection $a_{(i+1)i}$ is updated according to the contribution from node i to the output of node $i+1$ when event i is presented. Although the output of node i is not constant during period i, Eqn. 3.5 can still be used for training, because it provides the right direction to change the weight to minimize the error function (see Appendix A for detail).

When node i is being trained to recognize an event i, all the other nodes except node $i+1$ should be inactive. If another node is active, then the inhibitory input from that node to node i is strengthened slightly, using the same rule.

Training of the sequence nodes in the network is completed for one event, then for another, etc. in sequential order, by changing the weight of every interconnection in small increments. The training procedure is repeated after all the events in a sequence have been presented. This is performed until all the sequence nodes have been trained to keep track of the events in a sequence.

The training algorithm requires all the weights of the sequence nodes in the network to be modified in small increments simultaneously. Simulation results show that many iterations are required to train the proposed network. To speed up the training process, another training algorithm, which requires only some of the interconnect weights to be adjusted, is given in Chapter 4.

3.4.2. Result Node

The output from each sequence node is connected to the input of the result node R, by an excitatory connection having a fixed weight c_i (see Eqn. 3.1). After all the sequence nodes have been trained to respond correctly to a sequence, the result node will be trained such that it will only be activated after the correct number of events have been presented to the network. The result node has only one adjustable parameter, namely the node bias, θ_r. Time constant τ_r of the result node is fixed, and is defined by $n \times T$ where $n \leq N$ and N is the number of sequence nodes.

Before the training begins, the weight c_i is given a fixed value. An estimate of c_i can be obtained using the analysis given in Chapter 4. To train the result node, a temporal sequence is presented. At the end of the sequence, θ_r is updated based on the error of the output compared to the desired response. The bias is decreased slightly if the output of the result node reaches the level of 1 before the predefined number of correct events. It is increased slightly otherwise. This training procedure is repeated for many iterations until the result node has been trained.

3.5. Multi-Module Network

The network in Fig. 3.5 can be trained to recognize only one temporal sequence, and it will be named a *temporal module*. A network with several temporal modules can be used to recognize multiple sequences. The use of modules in a network allows the size of a network to scale up easily.

This section presents the modular organization of the proposed network and the training of the network to recognize multiple temporal sequences.

3.5.1. Architecture

A modular network is shown in Fig. 3.8. Two temporal modules are connected to the same external inputs via a common bus. The result node of each module receives inhibitory inputs from the result node of the other module, such that the two modules operate in a winner-take-all fashion. The module recognizing a correct sequence achieves the maximum output, while output of the other modules is inhibited.

The network shown in Fig. 3.8 has only two modules. However, because of the regular structure, the network can be easily extended to have more than two modules.

3.5.2. Training

When there are several modules in a network, the result node in each module receives inhibitory inputs from the result nodes of other modules. The weights of these inhibitory connections must be adjusted so that the modules work in a winner-take-all fashion. The module responding to the correct sequence should have an output of 1, and the other modules should have an output of 0.

Individual modules are trained separately to recognize their corresponding sequences. Then, they are put together and trained to operate in a winner-take-all fashion by adjusting the inhibitory weights between them. At the end of the presentation of a particular sequence, the module trained to respond to that sequence should be activated, and all the other modules should remain quiescent.

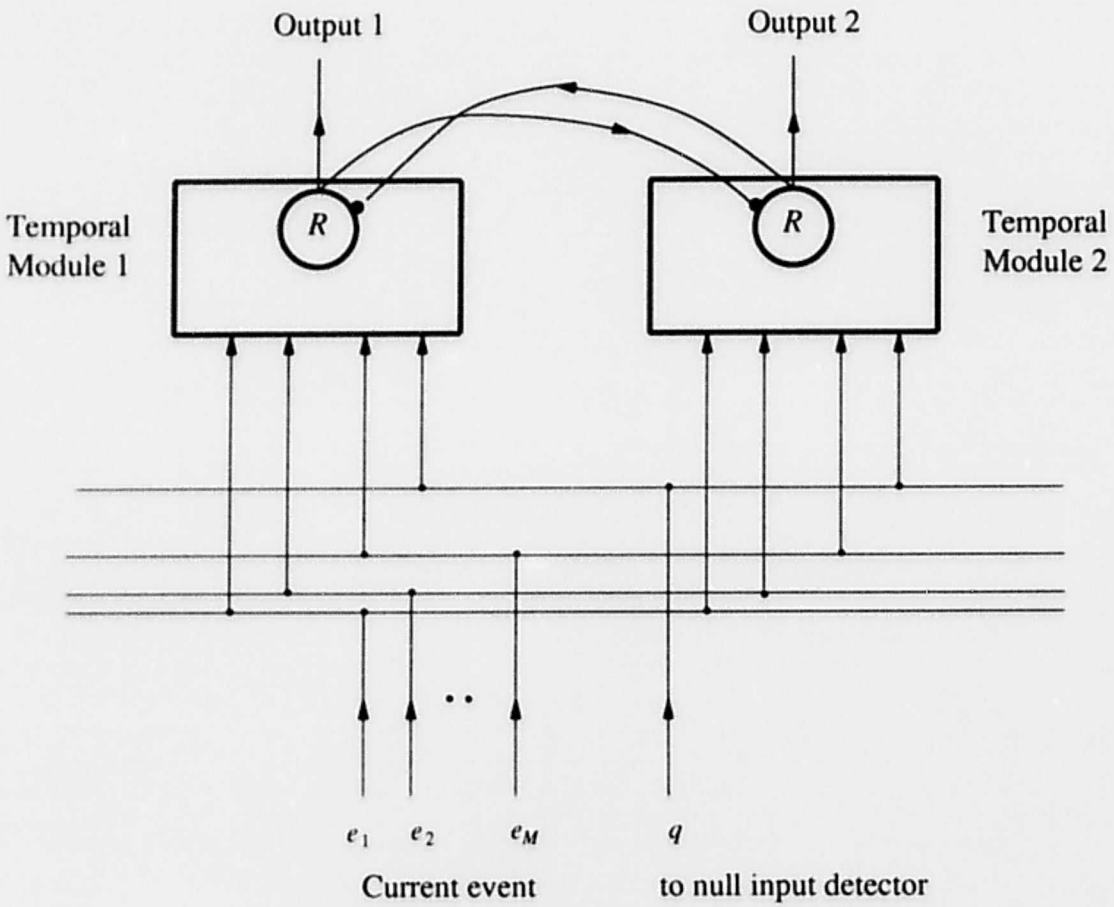


Fig. 3.8 A network with two temporal modules.

That is, the desired output of the winning module is 1 and the desired outputs of all other modules are 0. The inhibitory weights from the winning module to the other modules are increased slightly at the end of a sequence when the other modules are wrongly activated. Otherwise the weights are decreased.

Training of the modules is completed for one sequence, then for another, etc. After applying all the sequences, the training process is repeated until the network behaves in a winner-take-all fashion.

An alternative and easier approach for determining the inhibitory weights between modules will be presented in Chapter 4.

3.6. A Simple Example

To demonstrate the operation of the proposed network, several networks of different sizes were simulated and tested. The simulation results will be presented in Chapters 5 and 6. Here, a network having three sequence nodes and a result node will be used to illustrate the approach. The network was trained to recognize a sequence of three events, $I_s = \{E_1, E_2, E_3\}$, presented in successive sampling periods. The time constants of sequence and result nodes are $\tau_s = 10^{-4}$ and $\tau_r = 3\times10^{-4}$ second, and the sampling period is $T = 10^{-4}$ second. The following describes the response of the network when a correct sequence and a reverse sequence are presented.

The nodes' responses, when a correct sequence $t_{s1} = \{E_1, E_2, E_3\}$ is presented to the network, are given in Fig. 3.9. Since t_{s1} is the same as the training sequence, the difference $D(I_s, t_{s1})$ is zero. Each node responds with a fixed rise and decay time constant. The sequence nodes are activated one at a time, from left to right. Node S_1 is partially activated by event E_1, followed by full activation of subsequent nodes S_2 and S_3 by events E_2 and E_3, respectively. Node S_1 is partially activated because a node will be fully activated when it receives both the positive stimulus from its associated input event and a positive excitation from its upstream neighbor, as in the case of nodes S_2 and S_3.

The time integral of the sequence node outputs causes the result node to be activated, producing an output of 1 at time $t_0 + 3T$.

A reverse sequence $t_{s2} = \{E_3, E_2, E_1\}$ and $D(I_s, t_{s2}) > \theta_1$ is presented to the network, where θ_1 is the threshold above which the two sequences are regarded as incorrect. The network's response to sequence t_{s2} is given in Fig. 3.10. When event E_3 is presented to the network during the first period node S_3 is only partially activated, because node S_2 upstream is inactive. Due to the inhibitory inputs from node S_3, the partial activation of S_3 keeps nodes S_2 and S_1 near reset during

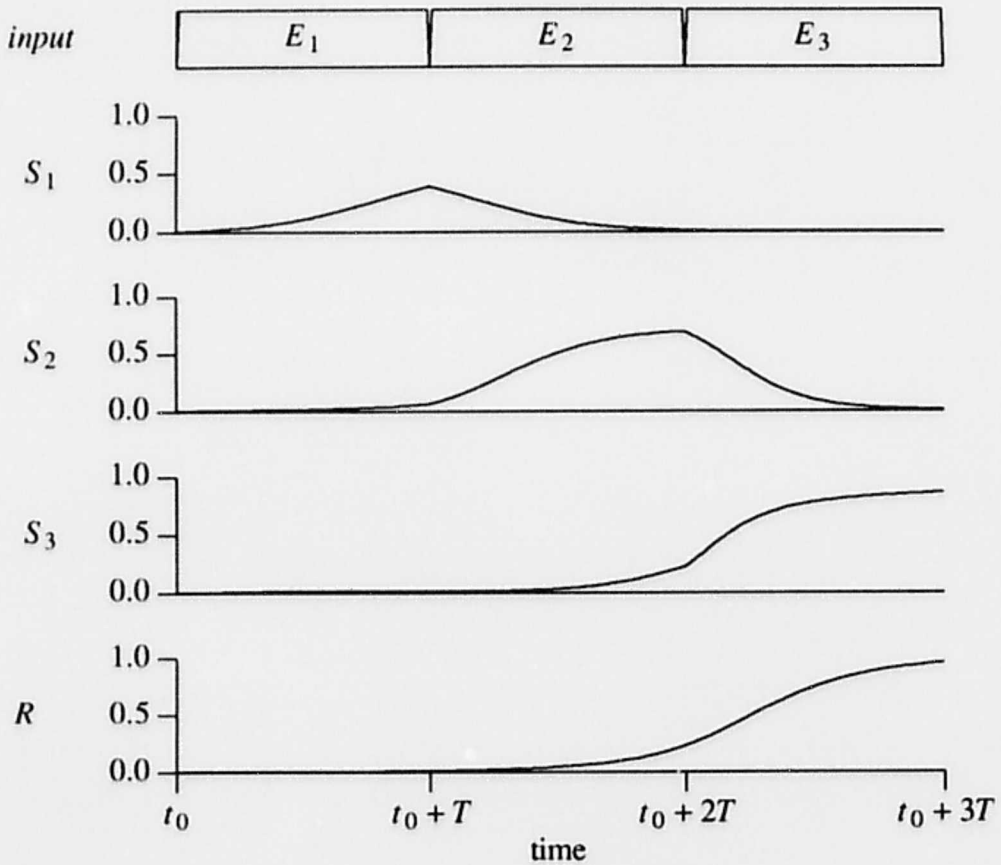


Fig. 3.9 Outputs of nodes S_1, S_2, S_3 and R when a correct sequence is presented.

the second and third periods, even though they receive their corresponding input events E_2 and E_1, respectively. The result node R is only slightly activated at the end of the whole sequence due to the lack of continuous excitation signals from the sequence nodes.

3.7. Network's Capability

This purpose of this section is to address the network's capability on the types of temporal sequence that the proposed network can be trained to recognize. To simplify the discussion, a sequence of events is treated as a string of symbols whose occurrences may be repeated in the

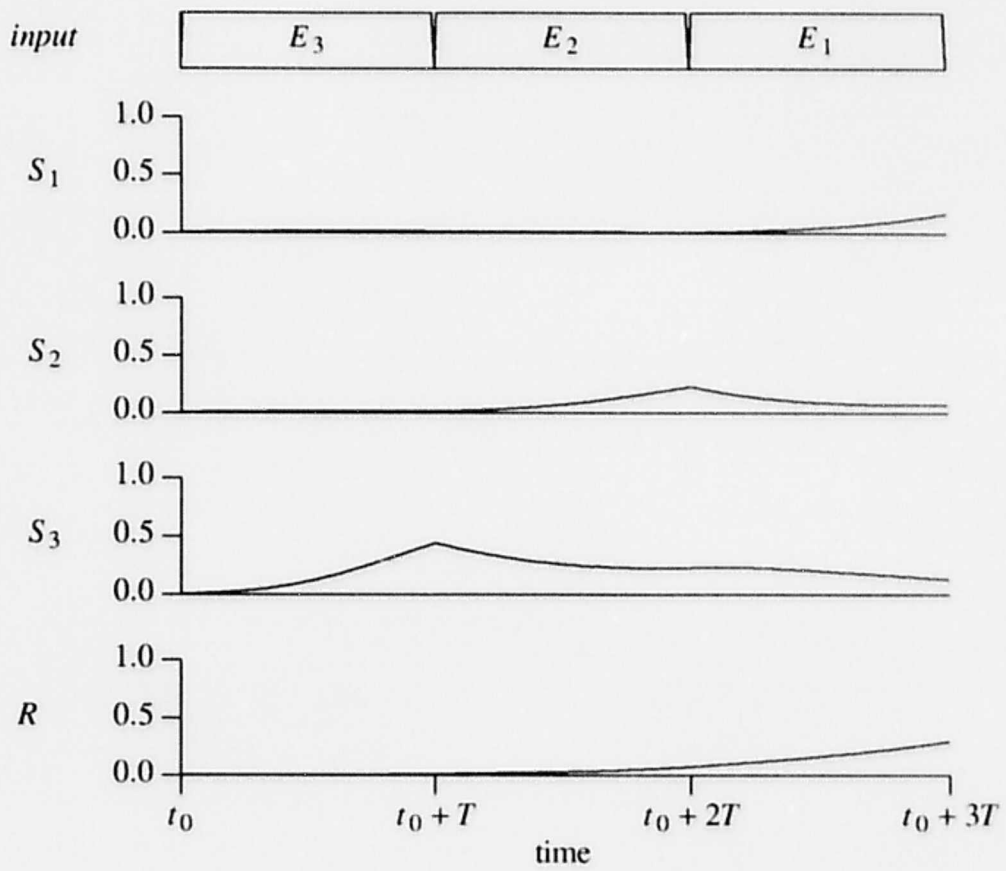


Fig. 3.10 Outputs of nodes S_1, S_2, S_3 and R when a reverse sequence is presented.

string and the elapsed time between events in a sequence is not taken into consideration.

Let $z_i \in Z$, $i = 1,2,...,U$, be the i*th* symbol in a set of symbols, Z, and U is the total number of symbols in the set. Strings are constructed from symbols by the operation of concatenation. For instance, the concatenation of two symbols z_1 and z_9, written as $z_1 z_9$, is the ordered sequence of the symbols. There are three possible ways that a string can be constructed. First, all symbols in a string are unique. An example is string $z_1 z_2 z_3 z_4$. Second, some symbols have one or more than one consecutive occurrences. An example is given by string $z_1 z_1 z_1 z_2 z_3$. And third, some symbols have one or more than one repeated occurrences anywhere in the string. An example is string

$z_1 z_2 z_1 z_3$ where symbol z_1 occurs twice in the first and third symbol.

With one memory element per state, the proposed network consisting of N sequence nodes has N active states (F_1, F_2, ..., F_N) and one reset state (F_R), as shown in Fig. 3.4. As a result, it is easy to train the network to recognize any string of length N as long as all N symbols in the string are different from each other and each symbol can be represented uniquely by one of the active network state. When such a string is presented, the network evolves from reset state F_R to active states F_1, F_2, F_3 and finally F_N. The time integral of these active states generates a result that represents the recognition of the string.

The unique representation of each symbol by one of the active states is no longer valid when there are repeated symbols in the string. The presentation of such symbol to the network causes two or more sequence nodes to be activated at the same time. Since each node is connected to other nodes, except for the node upstream, by inhibitory connections, the simultaneous activation of the sequence nodes causes their node outputs to be lower than one. The more sequence nodes are activated simultaneously, the lower their outputs. The time integral of these partially activated outputs causes the result node to generate a partial response. Therefore, strings with repeated symbols can cause the proposed network to be poorly trained.

3.8. Comparison With Other Neural Networks

As mentioned in section 3.2, it is possible to adapt both the time-delay and general sequential neural networks to have the same input-output behavior as the proposed network by adding a result node to the network to implement temporal integration. The main difficulty with this approach is the need for a complex and unwieldy pattern classifier. The presence of local minima in the error or energy function of a large and complex pattern classifier makes it difficult to train. A network with a complex pattern classifier is also difficult to analyze due to the lack of mathematical tools to analyze nonlinear systems.

The purpose of this section is to compare the enhanced time-delay and general sequential neural networks with the network proposed in this thesis, based on the required number of nodes, memory elements, and interconnect weights. The comparison is performed for a network having M inputs and for a temporal sequence consisting of N events. The required number of nodes, memory elements, and interconnect weights for each network will be expressed in terms of N and M.

3.8.1. Time-delay Neural Networks

In the time-delay neural network shown in Fig. 3.1, if all events in a sequence are stored before they are processed, then NM time delays or memory elements are needed.

Two types of pattern classifiers have been used in time-delay neural networks. The first is a multi-layer feed-forward neural network, which by far the most popular classifier. It is used in the networks proposed by [Watrous 1987, Lang 1988, Waibel 1988, Widrow 1988, Bengio 1989, Roitblat 1989, Bottou 1990, Hampshire 1990]. The second is a dynamic recurrent neural network. A recurrent neural network proposed by Hopfield [Hopfield 1982] was used by Tank and Hopfield as part of their time-delay neural network [Tank 1989].

The multi-layer feed-forward pattern classifier in Fig. 3.1 has N output nodes and several layers of hidden nodes. Let H_i be the number of nodes in the ith hidden layer. Then, the number of nodes in the network is the sum of the number of output nodes and hidden nodes in each layer, giving a total of $(\sum_{i=1}^{L} H_i + N)$ nodes, where L is the number of hidden layers. The number of interconnect weights between two layers is the product of the numbers of nodes in the two layers. That is, there are NMH_1 interconnections between the input and the first hidden layer, H_1H_2 interconnections between the second and third hidden layers, ... and NH_L interconnections between the last hidden layer and the output layer. Therefore, the network's space complexity can be summarized as follows:

$$\text{number of nodes} = \sum_{i=1}^{L} H_i + N$$

$$\text{number of memory elements} = MN \tag{3.6}$$

$$\text{number of interconnect weights} = NMH_1 + \sum_{i=2}^{L} H_i H_{i-1} + NH_L$$

To implement the pattern classifier with MN inputs and N outputs using the Hopfield network requires a total of N nodes, connected to each other by N^2 interconnections. Each node is also connected to the outputs of MN time delays by MN external interconnections. Then for N nodes, there are a total of MN^2 external interconnections. Thus,

$$\begin{aligned} &\text{number of nodes} = N \\ &\text{number of memory elements} = MN \\ &\text{number of interconnect weights} = N^2 + MN^2 \end{aligned} \tag{3.7}$$

Note that the Hopfield network does not have any hidden nodes.

3.8.2. General Sequential Neural Networks

A sequential neural network can be used to classify a sequence of events by representing the arrival of one input event after another in a sequence as an evolution of the network from one state to another. Temporal information of a sequence is stored as the state of a sequential network. Therefore, unlike a time-delay neural network, a sequential network needs between $\log_2 N$ and N memory elements. There are different organizations for a sequential neural network. We will consider a few here.

The sequential neural network in Fig. 3.2. proposed by Prager, Harrison and Fallside has $\log_2 N$ memory elements [Prager 1986]. The Boltzmann machine was used to implement the pattern classifier in their network. To add an integrator to the network, a pattern classifier has $(M + \log_2 N)$ inputs and $(N + \log_2 N)$ outputs. Let H be the number of hidden nodes in the Boltzmann machine. Then, the pattern classifier has $(N + \log_2 N)$ output nodes and H hidden nodes, yielding to a total $(H + N + \log_2 N)$ nodes. The nodes are connected to each other by $(H + N + \log_2 N)^2$ interconnections. Each node is also connected to each of the $(M + \log_2 N)$ inputs by an external

interconnection. To connect $(H + N + \log_2 N)$ nodes in the network to $(M + \log_2 N)$ inputs requires $(M + \log_2 N)(H + N + \log_2 N)$ external interconnections. Therefore, the space complexity of the network can be expressed as follows:

$$\begin{aligned} &\text{number of nodes} = H + \log_2 N + N \\ &\text{number of memory elements} = \log_2 N \\ &\text{number of interconnect weights} = (M + \log_2 N)(H + \log_2 N + N) + (H + \log_2 N + N)^2 \end{aligned} \tag{3.8}$$

Several networks which use a multi-layer feed-forward neural network to implement the pattern classifier in a sequential network were proposed [Jordan 1989, Elman 1990]. Memory elements that store the temporal information in a sequence are implemented by a layer of self-feedback context nodes, as shown in Fig. 3.11. The pattern classifier may have several layers of hidden nodes and one layer of output nodes. The figure shows a network with only one layer of hidden nodes. To add an integrator to the network, N output and N context nodes are required. By taking the context nodes as memory elements, then the multi-layer feed-forward classifier has N outputs and $M+N$ inputs. Using the same argument for deriving the number of nodes and interconnect weights required by the multi-layer feed-forward classifier presented in the previous section, the number of nodes, memory elements, and interconnect weights required by the model are as follows:

$$\begin{aligned} &\text{number of nodes} = \sum_{i=1}^{L} H_i + N \\ &\text{number of memory elements} = N \\ &\text{number of interconnect weights} = (M+N)H_1 + \sum_{i=2}^{L} H_i H_{i-1} + NH_L \end{aligned} \tag{3.9}$$

Not all sequential neural networks can be adapted to have the required input-output behavior specified in Chapter 1. One example is the network proposed by Stornetta, Hogg and Huberman [Stornetta 1987]. The network has a layer of self-feedback input nodes and a pattern classifier implemented by a multi-layer feed-forward network. It is difficult to modify the network to meet the required input-output requirement, because each input node is connected to only one input line