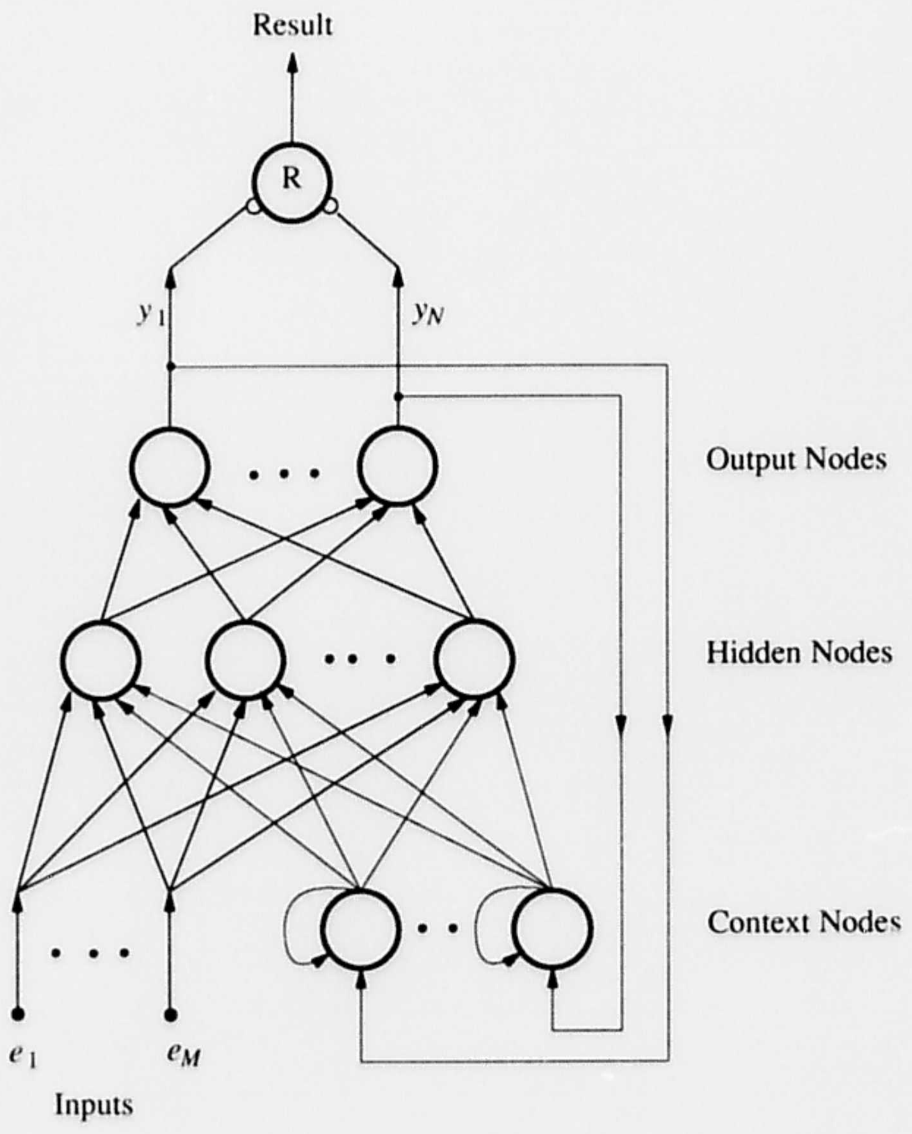


Fig. 3.11 A sequential neural network with the outputs of a multi-layer feed-forward pattern classifier connected to a result node *R*.

and thus cannot recognize an input event. The pattern classifier in that design is used mainly to recognize the time-dependent output pattern of all input nodes. The network proposed by Shamma also has the same limitation [Shamma 1989].

3.8.3. Proposed Network

In the proposed network, there are N sequence nodes, each of which containing a memory element. The sequence nodes are mutually connected to each other by N^2 interconnections. Each sequence node is also connected to M external inputs. Therefore, the number of nodes, memory elements, and interconnections can be summarized as follows:

$$\begin{aligned} \text{number of nodes} &= N \\ \text{number of memory elements} &= N \\ \text{number of interconnect weights} &= N^2 + NM \end{aligned} \tag{3.10}$$

Note that the proposed network does not require any hidden nodes.

3.8.4. Discussion

A set of equations were derived above for each type of network to estimate the number of nodes, memory elements, and interconnect weights as a function of the number of input lines, M, and the number of input events, N. For purpose of comparison, two different values of N and M are considered here, one representing a small network and the other representing a large network. Tables 3.1a and 3.1b show the estimated number of nodes, memory elements, and interconnect weights for each network model when $N = 16, M = 64$, and $N = 128, M = 256$, respectively. In the case of multi-layer networks, a single hidden layer containing $\frac{N}{2}$ nodes is assumed.

The tables show that both time-delay network models have an inherent problem with scaling. Many memory elements and interconnect weights are needed when N and M are large. The number of memory elements and interconnect weights required are in the order of MN and MN^2, respectively. The large number of memory elements required implies that a time-delay network is very expensive to implement. The large number of interconnect weights makes the network difficult to train. Training becomes more difficult in the presence of hidden nodes, which leads to a complex error function with many local minima. A large number of memory elements and interconnect weights also makes VLSI implementation of the network difficult, because there is a limitation to

Network models	Nodes	Memory elements	Interconnect weights	Hidden nodes
Time-delay multi-layer feed-forward network	24	1024	8320	Yes
Time-delay Hopfield network	16	1024	16640	No
Sequential network using a Boltzmann machine as a pattern classifier	28	4	2688	Yes
Sequential network using a multi-layer feed-forward network	24	16	688	Yes
Proposed network	16	16	1280	No

a) $N = 16$ and $M = 64$

Network models	Nodes	Memory elements	Interconnect weights	Hidden nodes
Time-delay multi-layer feed-forward network	192	32768	2105344	Yes
Time-delay Hopfield network	128	32768	4210688	No
Sequential network using a Boltzmann machine as a pattern classifier	199	7	91938	Yes
Sequential network using a multi-layer feed-forward network	192	128	32768	Yes
Proposed network	128	128	49152	No

b) $N = 128$ and $M = 256$

Table 3.1 Estimated number of nodes, memory elements, and interconnect weights required of each network model for different values of N and M.

the number of transistors that can be fabricated on a chip.

In contrast with time-delay neural networks, a general sequential neural network can be scaled up easily. The number of memory elements and interconnect weights required is in the order of N and N^2, respectively. However, a general sequential network is difficult to train and analyze because of the complex dynamical interactions between nodes. Hidden nodes are needed in most cases, causing the network to be even more difficult to train and analyze.

The proposed network uses one memory element per state and thus is a special case of a general sequential neural network. Therefore, like most general sequential network, it requires the same order of nodes and interconnect weights, and thus can be scaled up easily. The fact that the proposed network has only a single layer of nodes and all the intermediate states are known during a training session permits the use of a simple LMS training algorithm, even though the nodes are mutually connected. Since there are no hidden nodes, the error function is bowl shaped and has only a global minimum. Thus the network can be trained easily.

The assignment of a separate memory element for each state along the state trajectory that corresponds to a correct sequence means that the network has no more than one node active at any given time. It will be shown in Chapter 4 that the network can be analyzed easily, because it is sufficient to consider only one or two nodes at any given time in the analysis. The effect of other nodes can be ignored. This is contrary to most of the general sequential networks, which require the interactions between all nodes be taken into account in the analysis. This can be a complex task when the number of nodes in a network is large.

3.9. Concluding Remarks

A special configuration of a sequential neural network has been proposed for use in temporal pattern recognition. The proposed network uses one memory element per state to keep track of each event in a sequence. The recognition results of individual events are then summed in time to generate a final response. The network has a lower space complexity than the time-delay networks. It has a complexity similar to that of general sequential networks, but it is easier to train and analyze.

Chapter 4

Analysis

4.1. Introduction

This chapter provides an analysis of the behavior of the neural network proposed in this thesis to recognize a sequence of events in time. There are two purposes for performing this analysis. One is to understand why the proposed network is able to recognize a temporal sequence and the second is to provide insight into the working ranges of the interconnection weights between nodes. To know approximately the working values of the interconnection weights is vital in a VLSI implementation of the network because of the limited resolution available when constructing an interconnection with adjustable weight. Knowing the initial value of each interconnection weight also reduces the time required to train a network, as will be shown in Chapter 5.

This chapter begins by introducing a piecewise linear approximation used to analyze the dynamics of the temporal nodes. An analysis of the interactions between sequence nodes to trace the events in a given input sequence is presented, followed by an analysis of the temporal integration process performed by the result node in a temporal module. An analysis of the competition between the result nodes when there are many interconnected temporal modules is also presented. The analytical results to estimate the working range of each interconnection weight in a temporal module are given in each analysis. Finally, a simplified training algorithm based on the analytical results is presented.

4.2. Piecewise Linear Approximation

The proposed neural network is composed of many simple nodes interconnected by linear weighted links. The internal state of each node is the integral in time of the products of every weight and the corresponding output of other connected nodes (see Eqn. 3.4). In Chapter 3, the input-output relation of a node was assumed to be a sigmoidal transfer function of the form

$$y_j = \frac{1}{1+e^{-x_j}} \tag{4.1}$$

where x_j, y_j are the input and output of node j, respectively. For the purpose of analysis, piecewise linear approximation given below will be used. Two transfer functions are shown in Fig. 4.1.

$$y_j = \begin{cases} 0 & x_j \le -2.5 \\ 0.2x_j + 0.5 & -2.5 < x_j < 2.5 \\ 1 & x_j \ge 2.5 \end{cases} \tag{4.2}$$

The analysis presented in the rest of this chapter will concentrate on the linear region of the transfer function, i.e. where $-2.5 < x_j < 2.5$. This is the main region of interest while a node is changing state. The node will be active when its internal state is between 0 and 2.5, and inactive when its internal state is between 0 and -2.5 . Computer simulation of the network will use the sigmoidal function for comparison.

4.3. Dynamics of Sequence Nodes

When the events of a correct temporal sequence are presented at regular intervals to a temporal module, the sequence nodes will be activated one at a time. The activation sequence travels from left to right keeping track of the input events and the elapsed time between events.

Consider the sequence node on the left of the figure in Fig. 4.2. The weights of the external input connections for the node are configured or trained so that it can recognize only its associated event, E_i. For the purpose of analysis, the external connections will be approximated by a single

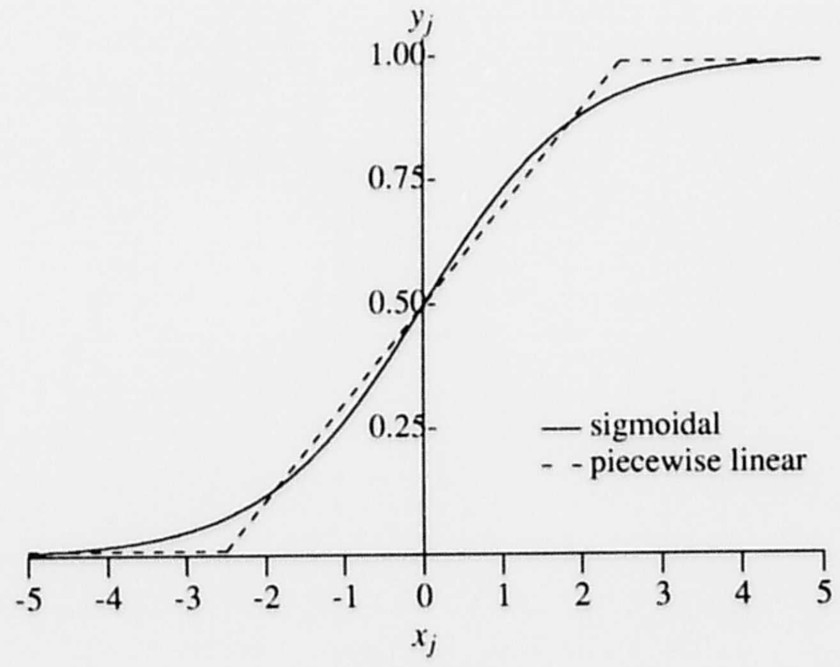


Fig. 4.1 Comparison between the sigmoidal and piecewise linear transfer functions.

input z_i. Event E_i will be represented by a unit step appearing on input z_i, which is otherwise equal to zero. Using this simplified model, the dynamics of the sequence nodes will now be analyzed.

Consider a simple network consisting of 3 sequence nodes interconnected as shown in Fig. 4.3. An input sequence to a module can be simulated by applying unit pulses on the external inputs z_1, z_2, and z_3 of the nodes. A unit pulse with a duration of one sampling period appearing on the external input z_i of node i implies that the event which should be recognized by the node is present at the input of the machine during that period. Hence, a correct input sequence is represented by the three pulses shown in Fig. 4.4. All other input sequences are treated as either partial or incorrect sequences. In the analysis, we want to show that the nodes in the figure will fire one at a time from left to right when a correct input sequence is presented. Otherwise, the sequence nodes remain reset most of the time.

4.3.1. Analytical Solution

When a correct sequence is presented to the module shown in Fig. 4.3, it is sufficient to analyze two sequence nodes at a time. Let us see why this is the case.

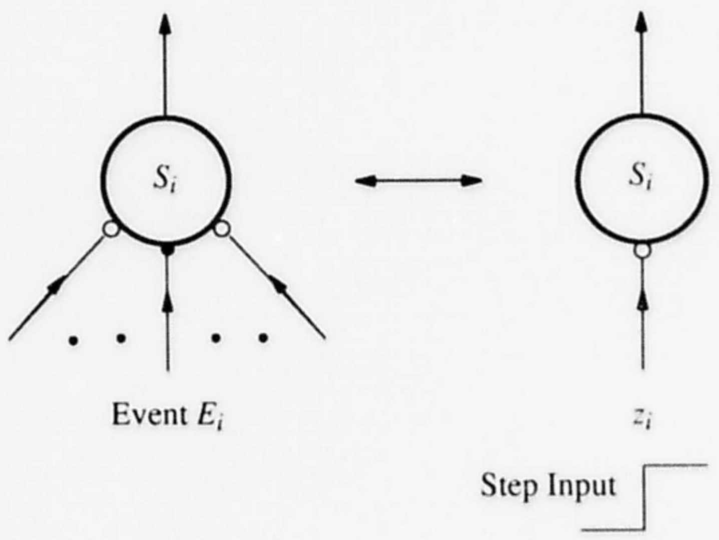
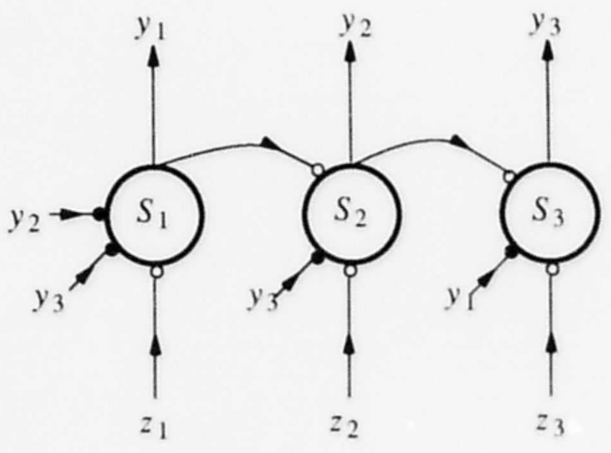


Fig. 4.2 Interpretation of external inputs to a node



○ excitatory connection
● inhibitory connection

Fig. 4.3 A linear chain of sequence nodes

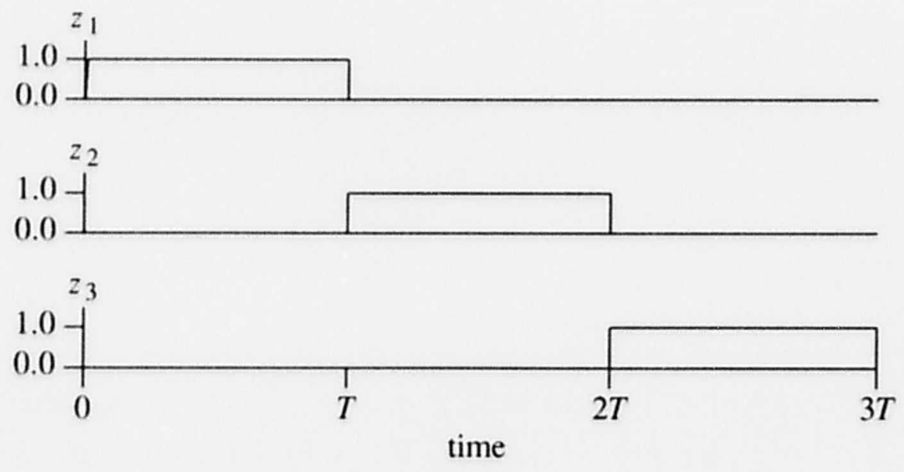


Fig. 4.4 A sequence representing a correct input sequence.

When a unit pulse is presented to the external input z_1 between $t = 0$ and $t = T$, node S_1 will be partially active because it is the first node, and hence, it does not receive any excitation from the other two nodes. During this period, node S_2 is partially activated by the excitation it receives from node S_1, and node S_3 is constantly reset by the inhibition from node S_1. Hence, only the two nodes S_1 and S_2 are active during period $0 \le t \le T$.

When the next unit pulse is presented to the external input z_2 between $t = T$ and $t = 2T$, node S_2 is fully activated, because it has been partially activated by node S_1 in the first period. Node S_1 is reset by the inhibition from node S_2 and node S_3 is partially activated by the excitation from node S_2. Node S_1 is required to be reset as soon as possible by node S_2, because of the inhibition from node S_1 to node S_3. Otherwise, node S_3 will not be sufficiently activated by node S_2. Therefore, as an approximation, the inhibition from node S_1 to node S_3 will be ignored in the analysis, and only nodes S_2 and S_3 will be treated as active during the period $T \le t \le 2T$. The appropriateness of this approximation will be verified in section 4.3.

Since there are at most two active sequence nodes at any time, analysis of two consecutive sequence nodes is sufficient to understand how a chain of sequence nodes can keep track of the events in a temporal sequence.

Consider two consecutive sequence nodes S_i and S_j. The dynamic equations for these two nodes can be expressed in the form

$$\tau_s \frac{dx_i}{dt} + x_i = bz_i - ay_j + \theta_s \tag{4.3}$$

$$\tau_s \frac{dx_j}{dt} + x_j = bz_j + ay_i + \theta_s \tag{4.4}$$

where τ_s is the time constant associated with each sequence node, b is the weight of the external input link, a is the magnitude of the weight of the link connecting the output of one node to the input of another node, z_k is the external input of node S_k, and θ_s is the internal bias in each node. The weights a, b and the internal bias θ_s are assumed to be the same for all nodes, because they all perform the same operation, namely, to recognize an input event and transmit the result to the next node. It has also been assumed that the excitatory and inhibitory connections between nodes have the same amplitude.

Consider the following input sequence shown in Fig. 4.5

$$z_i = h_i[u(t-(i-1)T) - u(t-iT)] \tag{4.5}$$

$$z_j = h_j[u(t-(j-1)T) - u(t-jT)] \tag{4.6}$$

where T is the input sampling interval, $u(t)$ is the unit step function, h_i and h_j are binary numbers taking the values of either 0 or 1.

Solving equations 4.3 and 4.4 yields the responses given below for nodes S_i and S_j when they are subjected to inputs z_i and z_j, respectively. The derivation of this solution can be found in

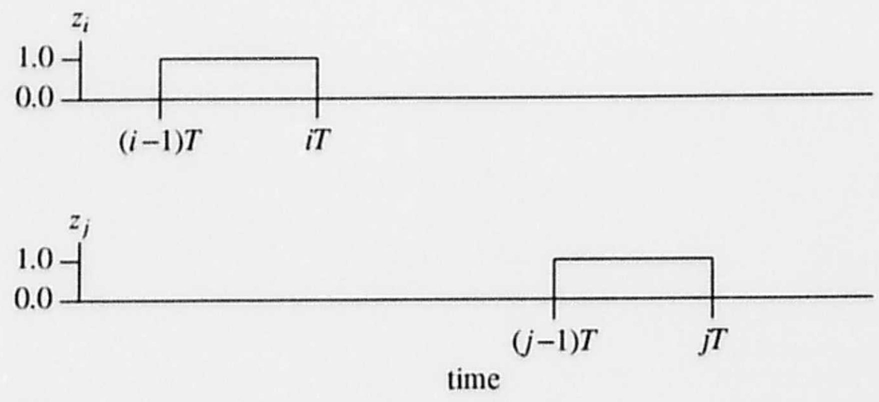


Fig. 4.5 An arbitrary input sequence.

Appendix B. Note that the following expressions gives the responses of nodes S_1 and S_2 when $i = 1$ and $j = 2$, or nodes S_2 and S_3 when $i = 2$ and $j = 3$.

The solution to the responses of nodes S_i and S_j will be used in the next few sections to analyze the temporal behavior of the sequence nodes in a module and also to estimate network parameters such as the time constant of a node.

$$\begin{aligned} x_i(t) &= e^{\frac{-t}{\tau_s}}[x_i(0)\cos\omega t - \frac{a}{5}x_j(0)\sin\omega t\,]u(t) \\ &+ \frac{[(\theta_s - \frac{a}{2})\beta_s(t) - \frac{a}{5}(\frac{a}{2} + \theta_s)\alpha_s(t)]u(t)}{1-a^2} \\ &+ \frac{bh_i[\beta_s(t-(i-1)T)u(t-(i-1)T) - \beta_s(t-iT)u(t-iT)]}{1-a^2} \\ &- \frac{abh_j[\alpha_s(t-(j-1)T)u(t-(j-1)T) - \alpha_s(t-jT)u(t-jT)]}{5(1-a^2)} \end{aligned} \tag{4.7}$$

$$\begin{aligned} x_j(t) &= e^{\frac{-t}{\tau_s}}[x_j(0)\cos\omega t + \frac{a}{5}x_i(0)\sin\omega t\,]u(t) \\ &+ \frac{[(\theta_s + \frac{a}{2})\beta_s(t) + \frac{a}{5}(\frac{a}{2} - \theta_s)\alpha_s(t)]u(t)}{1-a^2} \\ &+ \frac{bh_j[\beta_s(t-(j-1)T)u(t-(j-1)T) - \beta_s(t-jT)u(t-jT)]}{1-a^2} \\ &+ \frac{abh_i[\alpha_s(t-(i-1)T)u(t-(i-1)T) - \alpha_s(t-iT)u(t-iT)]}{5(1-a^2)} \end{aligned} \tag{4.8}$$

where

$$\beta_s(t) = 1 - e^{\frac{-t}{\tau_s}}\cos\omega t - \tau_s\omega e^{\frac{-t}{\tau_s}}\sin\omega t \tag{4.9a}$$

$$\alpha_s(t) = 1 - e^{\frac{-t}{\tau_s}}\cos\omega t - \frac{1}{\tau_s}\omega e^{\frac{-t}{\tau_s}}\sin\omega t \tag{4.9b}$$

$$\omega = \frac{a}{\tau_s} \tag{4.9c}$$

4.3.2. Node Responses

First, we want to examine the interaction between two consecutive nodes when node S_i is active while the other node S_j, (where $j = i + 1$), is inactive without any influence from the external inputs; that is, $h_i = 0$, $h_j = 0$, $x_i(0) = 2.5$ and $x_j(0) = -2.5$. The responses of S_i and S_j when both nodes are operating in the linear region are shown in Fig. 4.6. The weights of the interconnections are $b = 1$, $a = 0.5$, $\theta_s = -0.1$ and $T = \tau_s$.

The solid lines in the figure are the results obtained from the analytical solutions using piecewise linearization, and the dotted lines are the results obtained from simulation using the sigmoidal transfer function. The figure indicates that S_j is partially activated by the excitation from S_i, and at the same time, S_i is being slowly turned off by the inhibition from S_j. The figure also shows that results obtained by applying piecewise linearization of the sigmoidal transfer function are close to the results obtained from computer simulation using the sigmoidal transfer function.

When external inputs are applied to the nodes, there are two cases of interest. The first case is that of the start-up response when nodes S_i and S_j correspond to either the first two sequence nodes in the module that have encountered the beginning of a sequence, or to any two consecutive sequence nodes in the chain that have encountered the beginning of a partial sequence. The other case is the steady state response of the sequence nodes after a number of events in a sequence have been recognized by the first few nodes in a module.

The first case is an important one because it represents the responses of the nodes when the first two events in a sequence are presented. We shall consider the analysis of the nodes' responses when S_i and S_j correspond to nodes S_1 and S_2 in Fig. 4.3. Initial, the nodes are both reset, i.e. $x_1(0) = -2.5$ and $x_2(0) = -2.5$. They are subjected to external inputs given by Eqns. 4.5 and 4.6, where $h_1 = h_2 = 1$, which represent the first two events in a correct sequence. The solid and dotted lines in Fig. 4.7 show the nodes' responses obtained from analytical solutions and computer simulation, respectively. Node S_1 is partially activated in response to an input pulse between $t = 0$ and $t = T$, and at the same time, helps S_2 to be partially activated. Between $t = T$ and $t = 2T$, node

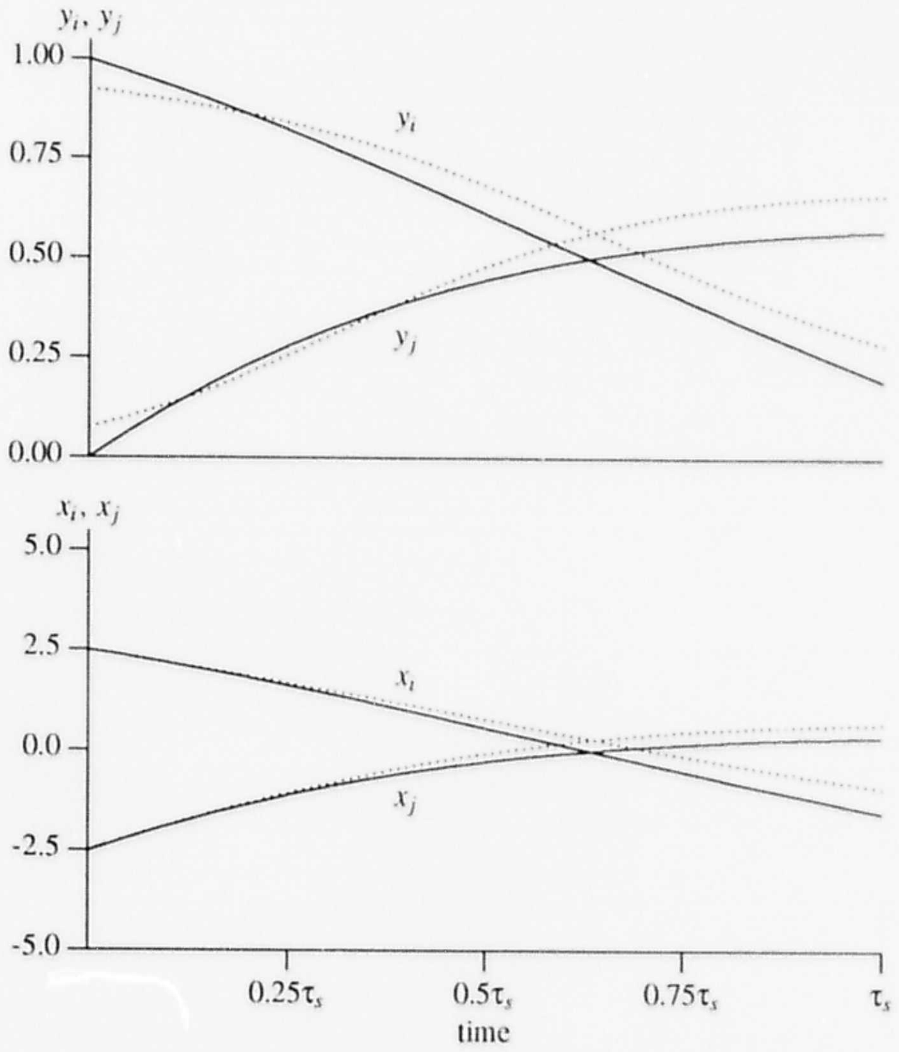


Fig. 4.6 Responses of nodes S_i and S_j when $h_i = 0$ and $h_j = 0$. Solid lines are the result obtained from an analytical solution, and dotted lines from simulation.

S_2 is fully activated by its input pulse while node S_1 is reset by the inhibition from node S_2.

At time $2T$, the start up phase has been completed, because node S_2 has been fully activated. That is, after the first two events in a correct sequence have been presented to a module, the sequence nodes reach the steady state response, in which the nodes are fully activated from left to right, one node at a time. In the steady state response, each node in the chain is partially activated by its upstream neighbor, and also receives excitation from the external input. Thus, the activities in nodes S_2 and S_3 in Fig. 4.3 can be regarded as steady state. The analysis is performed by substituting the initial conditions $x_2(0) = 0.0$ and $x_3(0) = -2.5$ into Eqns. 4.7 and 4.8. The external inputs have $h_2 = h_3 = 1$. Fig. 4.8 shows that the nodes are fully activated, one at a time, by their respective external inputs.

Up to this point, a simple network consisting of 3 sequence nodes was analyzed by assuming that one of the nodes is not activated. Next, we want to verify the appropriateness of this assumption by examining the responses of all three nodes when they are fully interconnected. The responses of the sequence nodes in Fig. 4.3 to a correct sequence and to a reverse sequence will be presented. These results are based on numerical simulation of the network since it is too complex to obtain an analytical solution for the responses. Network parameters that are the same as above are used in the simulation.

Let us examine the responses of the sequence nodes in Fig. 4.3 when a correct sequence is presented to the network. The external inputs are $z_1 = u(t) - u(t-T)$, $z_2 = u(t-T) - u(t-2T)$, and $z_3 = u(t-2T) - u(t-3T)$. The internal states of all the nodes are all reset to -2.5 initially. The internal states of the nodes are given in Fig. 4.9. The response of the network for a complete correct sequence as obtained by simulation is shown in Fig. 4.9. Node S_1 is partially activated by the first input pulse z_1, followed by the full activation of subsequent nodes S_2 and S_3 by input pulses z_2 and z_3, respectively. The full activation of nodes S_2 and S_3 represents the steady state response of any two consecutive nodes in a chain. Hence, the presentation of a correct sequence causes the sequence nodes to be activated one at a time, from left to right.

The nodes' responses to a reverse sequence are given in Fig. 4.10. When the first input pulse z_3 is presented to the network, node S_3 is partially activated. Then, when their input pulses z_1 and z_2 are presented, nodes S_1 and S_2 are only partially activated, because they are constantly inhibited by node S_3. Hence, with an incorrect sequence most nodes remain near their reset state.

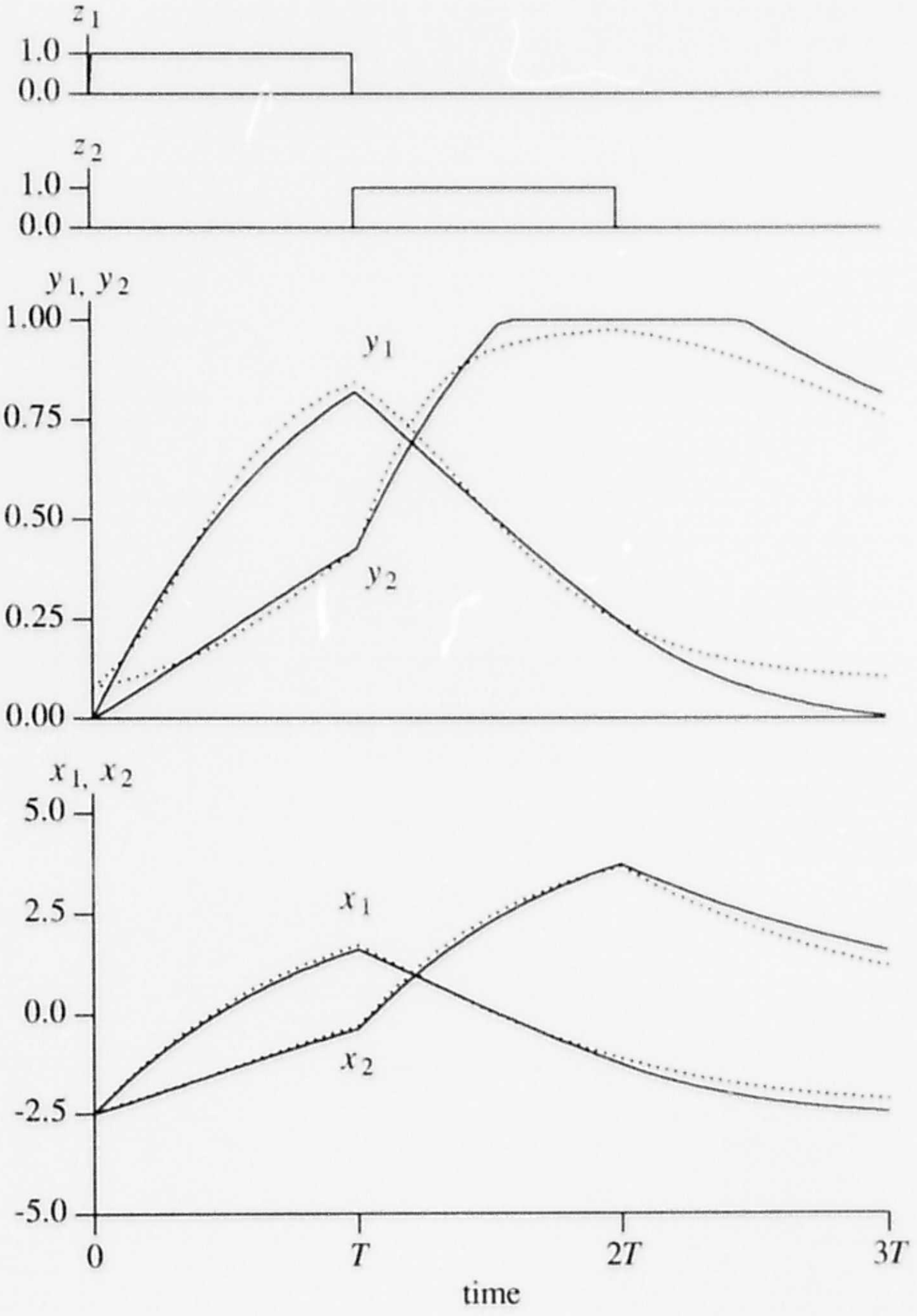


Fig. 4.7 Responses of nodes S_1 and S_2 when $z_1 = u(t) - u(t-T)$ and $z_2 = u(t-T) - u(t-2T)$. Solid lines are the results obtained from an analytical solution, and dotted lines from simulation.

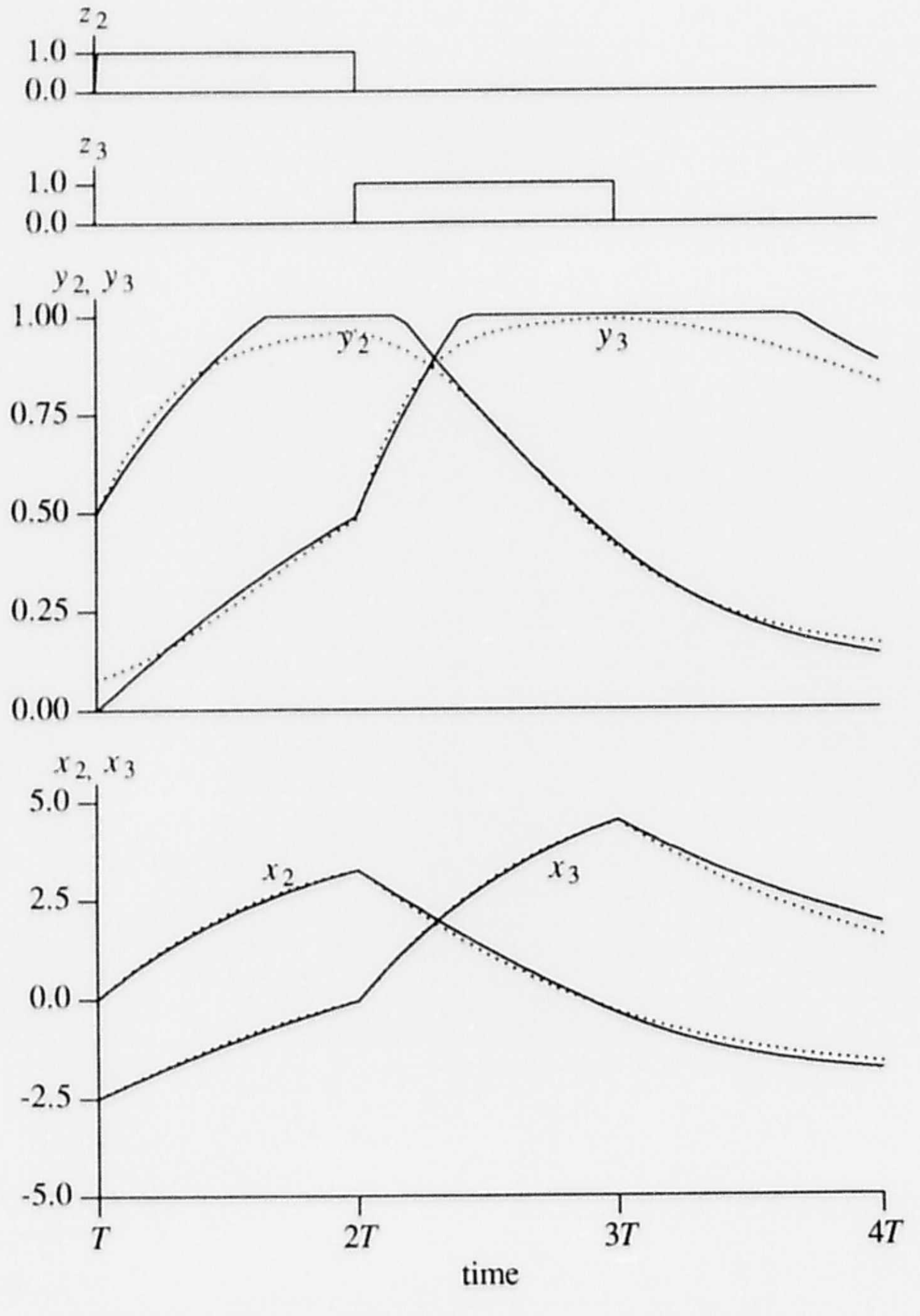


Fig. 4.8 Responses of nodes S_2 and S_3 when $z_2 = u(t-T) - u(t-2T)$ and $z_3 = u(t-2T) - u(t-3T)$. Solid lines are the results obtained from an analytical solution, and dotted lines from simulation.

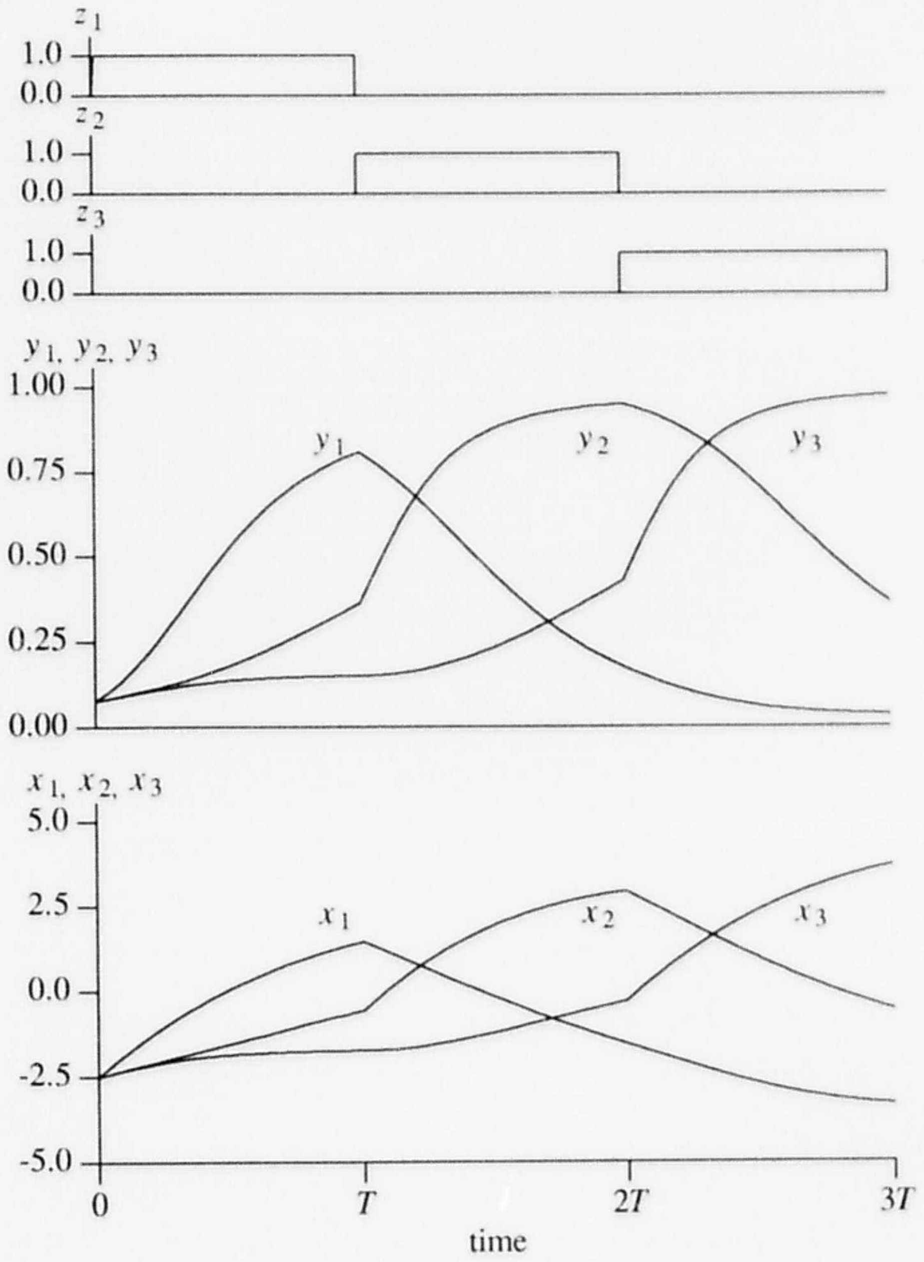


Fig. 4.9 Responses of nodes when $z_1 = u(t) - u(t-T)$, $z_2 = u(t-T) - u(t-2T)$ and $z_3 = u(t-2T) - u(t-3T)$ are presented.

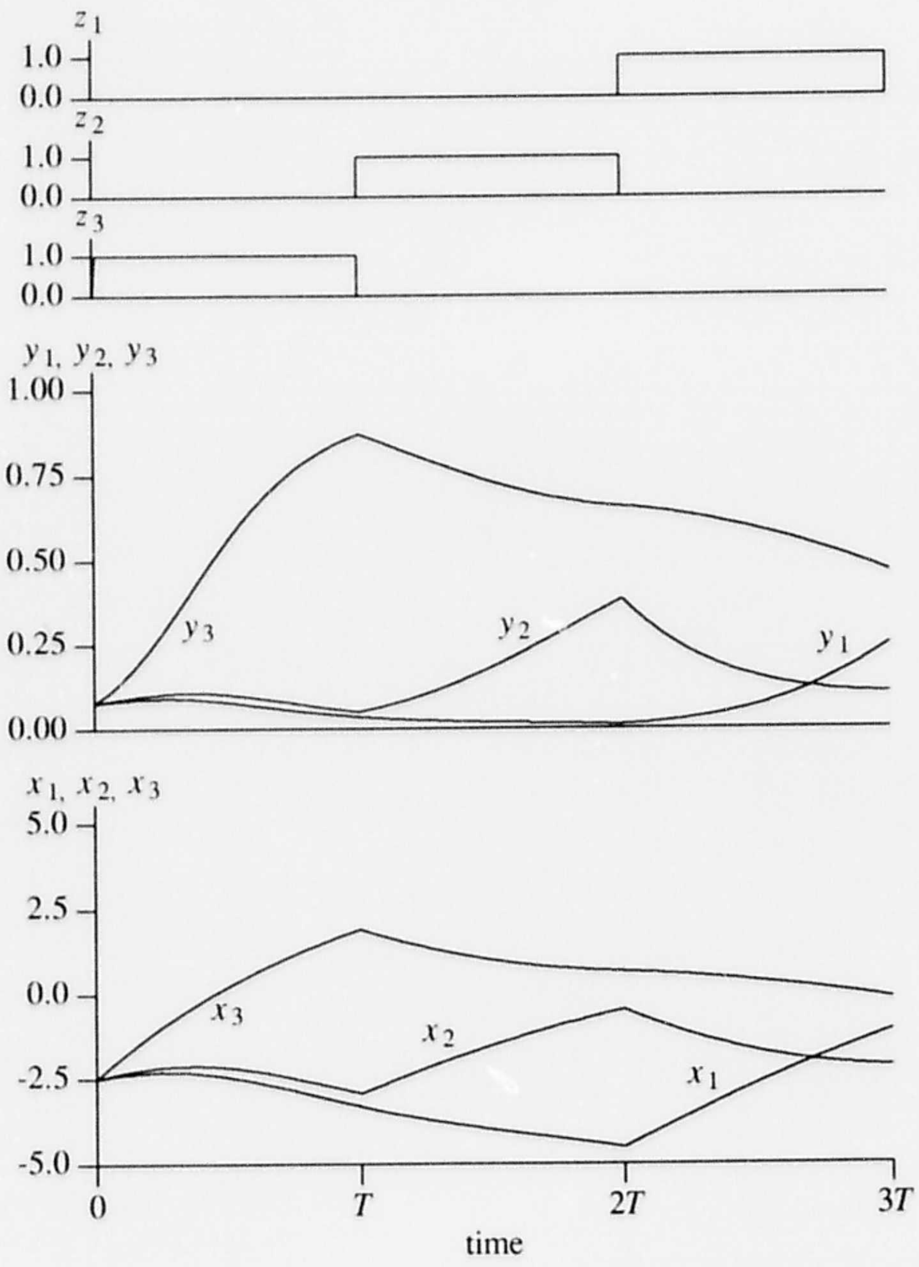


Fig. 4.10 Responses of nodes when $z_3 = u(t) - u(t-T)$, $z_2 = u(t-T) - u(t-2T)$ and $z_1 = u(t-2T) - u(t-3T)$ are presented.

4.3.3. Estimation of Sampling Period

In the proposed network, time-varying inputs are sampled at fixed time intervals of length T, generating a sequence of temporally related events. An important issue is the selection of a suitable sampling period for a given network. Alternatively, we need to determine the network parameters such as the time constant of the sequence nodes when the sampling period of a temporal recognition problem is given.

There are two constraints that determine the sampling rate. First, the period needed to activate a node is governed by its time constant, τ_s. Second, a finite time is required for an active node to cause the node downstream to be partially activated. As a result, the sampling period of a network can be estimated by making sure that during that period a node can be partially activated by its external input and the node downstream is only partially activated.

Consider now nodes S_1 and S_2 in Fig. 4.3. When an input pulse of duration T is presented to node S_1, the internal state $x_1(t)$ of node S_1 should rise from an initial value of -2.5 to a value slightly greater than 0 at the end of a sampling period, while the internal state $x_2(t)$ of node S_2 should rise from -2.5 to a value slightly less than 0. These inequalities can be expressed in the following.

$$0 \le x_1(T) \tag{4.10}$$

$$0 \ge x_2(T) \tag{4.11}$$

Substituting $x_1(t)$ and $x_2(t)$ given by Eqns. 4.7 and 4.8, respectively, for $x_1(0) = -2.5$, $x_2(t) = -2.5$, $h_1 = 1$, $h_2 = 0$, and $t = T$ yields the following expressions.

$$\begin{aligned} x_1(T) = {} & \frac{(\theta_s - \frac{a}{2})\beta_s(T) - \frac{a}{5}(\frac{a}{2} + \theta_s)\alpha_s(T)}{1-a^2} + \frac{b(\beta_s(0) - \beta_s(T))}{1-a^2} \\ & - 2.5e^{\frac{-T}{\tau_s}}(\cos\omega T - \frac{a}{5}\sin\omega T) \end{aligned} \tag{4.12}$$

$$\begin{aligned} x_2(T) = {} & \frac{(\theta_s + \frac{a}{2})\beta_s(T) + \frac{a}{5}(\frac{a}{2} - \theta_s)\alpha_s(T)}{1-a^2} + \frac{ab(\alpha_s(T) - \alpha_s(0)}{5(1-a^2)} \\ & - 2.5e^{\frac{-T}{\tau_s}}(\cos\omega T + \frac{a}{5}\sin\omega T) \end{aligned} \tag{4.13}$$

where β_s and α_s are the expressions given in Eqns. 4.9.

For example, consider a chain of sequence nodes with $\tau_s = 10^{-4}$, $a = 1$, $b = 1$ and $\theta_s = -0.1$. A plot of the values of $x_1(T)$ and $x_2(T)$ versus the sampling period T is shown in Fig. 4.11. In order to satisfy the inequality in both Eqns. 4.10 and 4.11, the samp ng period T should be in the range $0.55\tau_s < T < 0.80\tau_s$.

4.3.4. Estimation of Mutual Interconnect Weight

Consider now the range of values for the mutual interconnect weights, a, that meet the functional requirements of the sequence nodes when the values of τ_s, b and θ_s are given. To perform the analysis, again consider nodes S_1 and S_2 in Fig. 4.3, with the initial conditions $x_1(0) = 2.5$ and $x_2(0) = -2.5$.

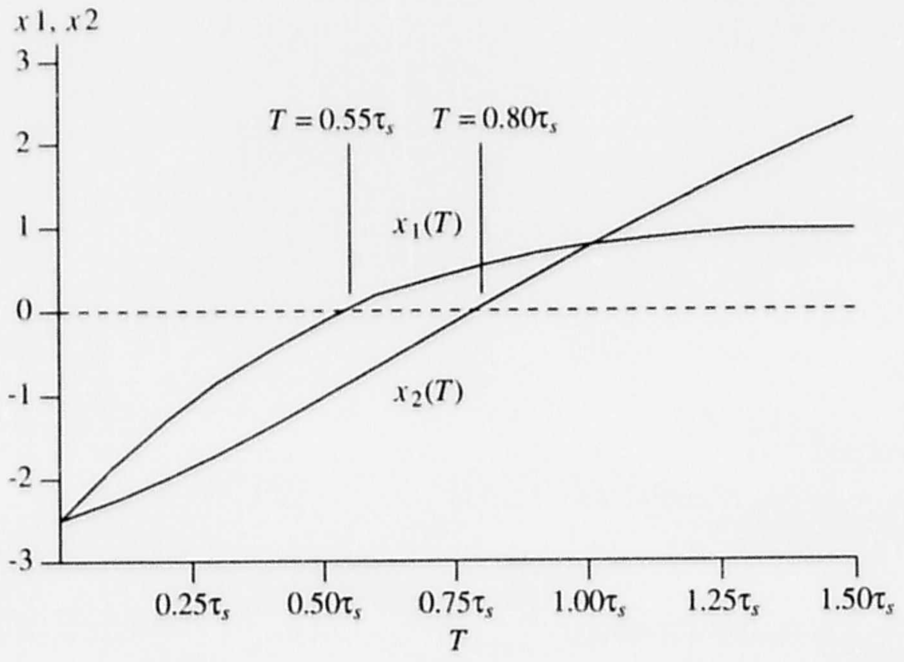


Fig. 4.11 Plot of $x_1(T)$ and $x_2(T)$ versus T in terms of τ_s.

Under no excitation from the external inputs, we are interested in how strong the excitatory connection from S_1 to S_2 should be for S_1 to provide sufficient excitation for S_2 to be partially turned on in one sampling period. Similarly, we would like to estimate the strength of the inhibitory connection from S_2 to S_1 in order to turn S_1 off in one sampling period.

The excitation from one sequence node to its downstream neighbor will be considered sufficient when this excitation alone causes the internal state of the node downstream to rise from an initial value of -2.5 to a value greater than 0 in one sampling period. For a node to be fully triggered, it must receive excitation from the node upstream and also from the external input when a correct event is received. Inhibition from one sequence node to the node upstream should be strong enough to cause the internal state of the latter to decay from an initial value of 2.5 to a value just below 0 in one sampling period.

Substituting the values of initial conditions and external inputs into Eqns. 4.7 and 4.8, the following conditions are necessary for proper operation.

$$\frac{(\theta_s - \frac{a}{2})\beta_s(T) - \frac{a}{5}(\theta_s + \frac{a}{2})\alpha_s(T)}{1-a^2} + 2.5e^{-1}(\cos\frac{aT}{\tau_s} + \frac{a}{5}\sin\frac{aT}{\tau_s}) < 0 \tag{4.14}$$

$$\frac{(\theta_s + \frac{a}{2})\beta_s(T) + \frac{a}{5}(\theta_s - \frac{a}{2})\alpha_s(T)}{1-a^2} + 2.5e^{-1}(\frac{a}{5}\sin\frac{aT}{\tau_s} - \cos\frac{aT}{\tau_s}) > 0 \tag{4.15}$$

where β_s and α_s are the expressions given in Eqns. 4.9 when $t = T$. The above conditions give us the first order approximation of the weight a of the connections between two sequence nodes as a function of τ_s, T and θ_s.

The conditions given in Eqns. 4.14 and 4.15 determine the acceptable range of the values of a. As an example, for $\tau_s = T = 10^{-4}$ and $\theta_s = -0.1$, we obtain $0.5 \le a \le 1.5$. For comparison, Fig. 4.12 shows a plot of the time needed as obtained by simulation for x_1 to drop from 2.5 to a value below 0 and for x_2 to rise from -2.5 to a value above 0 for different values of a. The initial conditions used in the simulation were $x_1(0) = 5$ and $x_2 = -5$. The dotted line $t = \tau_s$ indicates that for the nodes to be activated in one sampling period of τ_s seconds, the weights should be such that $0.75 \leq a \leq 1.35$.

4.3.5. Estimation of Input Connect Weight

Next let us determine a suitable range for the value of the weight b of the external inputs. Without encouragement from the sequence node upstream, the weight b should be chosen in such a way that the external input would cause the node to be only partially triggered in one sampling period T, i.e. the node's internal state should rise from -2.5 to 0. The internal state x_1 of a single node is given by

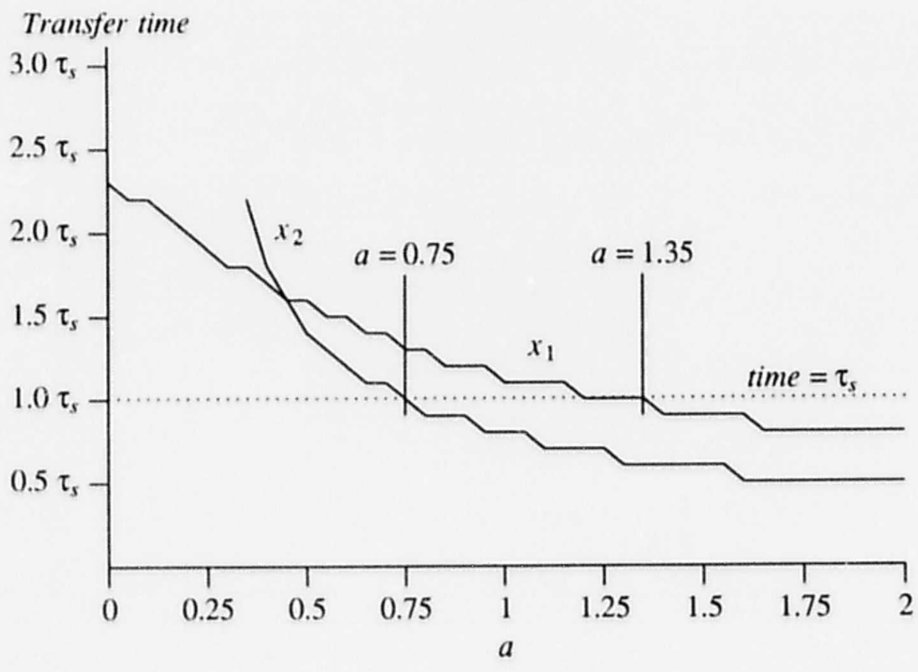


Fig. 4.12 Plot of the time required for internal states x_1 and x_2 to fall from 2.5 to a value less than 0 and rise from -2.5 to a value greater than 0, respectively, versus a.

$$\tau_s \frac{dx}{dt} + x = bz + \theta_s \tag{4.16}$$

where z is the external input and θ_s is the internal bias of the node. By solving and then substituting $z = 1$, $x(0) = -2.5$ and $x(T) = 0$, we obtain the following relation for estimating b:

$$b = \frac{5e^{\frac{-T}{\tau_s}}}{2\left(1 - e^{\frac{-T}{\tau_s}}\right)} - \theta_s \tag{4.17}$$

4.3.6. Summary

A simple analytical model has been developed for the interaction between the sequence nodes in a module. For a given application in temporal pattern recognition, the analysis provides a systematic approach for estimation of the network parameters.

The analytical results obtained from a piecewise linearization of the node's transfer function are close to the results obtained from computer simulation using a sigmoidal transfer function. This is an important observation, especially when the proposed network is considered for digital implementation on a DSP chip (see Section 7.2.2 for reference), because the piecewise linear transfer function is much faster to compute.

4.4. Dynamics of Result Nodes

The result node is responsible for integrating the outputs of the sequence nodes to determine whether a correct sequence has been received. When a temporal network has to recognize several input sequences, a separate temporal module is provided for each sequence. The result node in each module receives inhibitory inputs from the result nodes of other temporal modules, and the modules operate in a winner-take-all fashion. The module recognizing a correct sequence achieves the maximum output while outputs of all other modules are inhibited.

Analysis of the temporal integration process in the result node of one module is given in the next section, followed by the analysis for a network containing multiple modules.

4.4.1. Temporal Integration

Consider a temporal module consisting of N sequence nodes whose outputs are connected to the inputs of the result node by excitatory connections. An example for $N = 3$ is shown in Fig. 4.13. The result node consists of a leaky integrator that integrates the outputs of the sequence nodes with respect to time, with a time constant τ_r. Let the internal state of the result node be represented by a variable x_r, which satisfies the relation:

$$\tau_r \frac{dx_r}{dt} + x_r = \sum_{j=1}^{N} c y_j + \theta_r \tag{4.18}$$

where N is the number of sequence nodes, y_j is the output of sequence node j, c is the weight of the excitatory connection from any sequence node to the result node, and θ_r is the internal bias in the

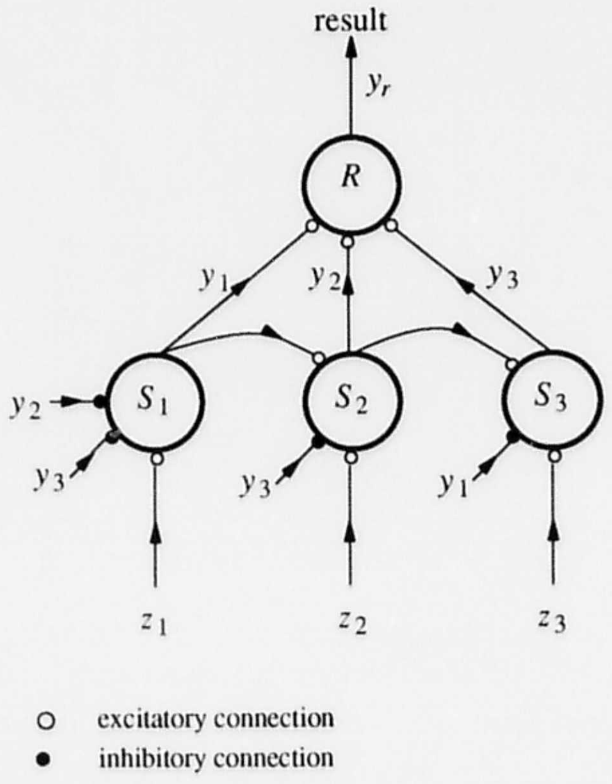


Fig. 4.13 A result node connected to a chain of sequence nodes

result node. The solution to this equation gives the behaviour of the result node for various inputs, as explained below.

4.4.1.1. Node Response to a Correct Sequence

The presentation of a correct sequence to a chain of sequence nodes causes them to be activated in sequence from left to right. The activities of the sequence nodes can be approximated by applying a pulse $u(t-(j-1)T) - u(t-jT)$ to the jth input line of the result node, where T is the input sampling period. Solving Eqn. 4.18 yields

$$\begin{aligned} x_r(t) &= c\sum_{j=1}^{N}((1-e^{\frac{-(t-(j-1)T)}{\tau_r}})u(t-(j-1)T) - (1-e^{\frac{-(t-jT)}{\tau_r}})u(t-jT)) \\ &+ \theta_r(1-e^{\frac{-t}{\tau_r}})u(t) + x_r(0)e^{\frac{-t}{\tau_r}}u(t) \end{aligned} \tag{4.19}$$

where $x_r(0)$ is the initial state of the result node.

Fig. 4.14 shows the responses of a result node integrating in time the simulated activities of 10 sequence nodes when they are subjected to a correct sequence. The parameters used are $T = 10^{-4}$, $\tau_r = 8T$ and $c = 1$. The solid lines show the output and the internal state of the result node when $\theta_r = -0.5$. The output rises from 0 to 1 and the internal state rises from -2.5 to 2.5, after all 10 events have been presented. The dotted lines are the responses when $\theta_r = -1.0$. In this case, the node is only partially activated. Thus, by varying the value of the internal bias, the output of a result node can be controlled.

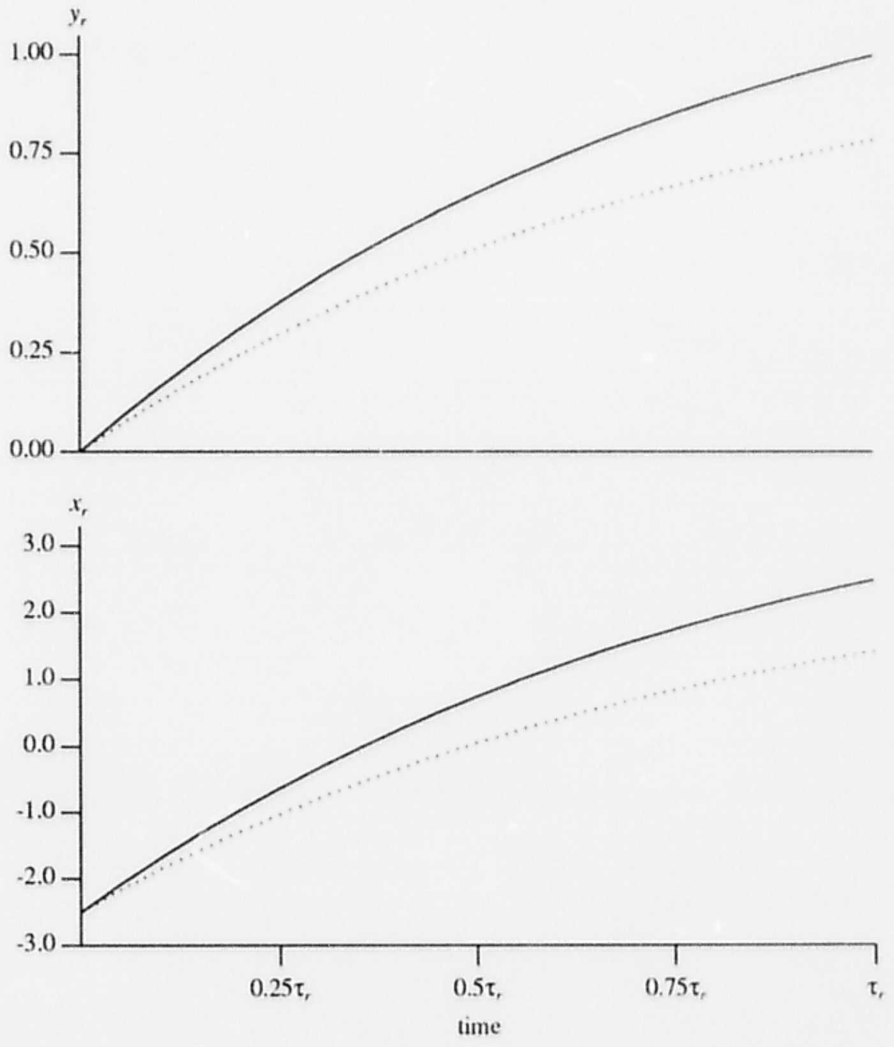


Fig. 4.14 Output and internal state of the result node when a correct sequence is applied. Dotted lines are the results using higher negative bias.

4.4.1.2. Estimation of Integration Time Constant

For a network consisting of N sequence nodes, we want to estimate a suitable time constant τ_r for the result node in relation to the input sampling period, such that the node will only be activated after a sufficient number of correct events have been presented.

Let N_{off} be the maximum number of events for which the result node will remain quiescent, and let N_{on} be the minimum number of events needed to activate the result node. The result node will be regarded as active when its internal state, x_r, is greater than 0. When $x_r < 0$, the node will be considered inactive. Substituting N_{off} and N_{on} and $x_r(0) = -2.5$ into Eqn. 4.19 gives the following two conditions for estimation of τ_r.

$$
\begin{aligned}
&c\sum_{j=1}^{N_{off}}\left(\left(1 - e^{\frac{-(t-(j-1)T)}{\tau_r}}\right)u(t-(j-1)T) - \left(1 - e^{\frac{-(t-jT)}{\tau_r}}\right)u(t-jT)\right) \\
&\quad + \theta_r\left(1 - e^{\frac{-t}{\tau_r}}\right)u(t) - 2.5e^{\frac{-t}{\tau_r}}u(t) < 0
\end{aligned}
\tag{4.20}
$$

$$
\begin{aligned}
&c\sum_{j=1}^{N_{on}}\left(\left(1 - e^{\frac{-(t-(j-1)T)}{\tau_r}}\right)u(t-(j-1)T) - \left(1 - e^{\frac{-(t-jT)}{\tau_r}}\right)u(t-jT)\right) \\
&\quad + \theta_r\left(1 - e^{\frac{-t}{\tau_r}}\right)u(t) - 2.5e^{\frac{-t}{\tau_r}}u(t) > 0
\end{aligned}
\tag{4.21}
$$

Given N_{off} and N_{on} as part of the design criteria and using Eqns. 4.20 and 4.21, it is possible to estimate a suitable value for τ_r. For example, assume that the required values of N_{off} and N_{on} are 4 and 6, respectively, for a network with the parameters $T = 10^{-4}$, $c = 1.0$ and $\theta_r = -0.1$, the values of the internal state, x_r, at $t = N_{on}T$ and at $t = N_{off}T$ are computed for different values of τ_r and plotted in Fig. 4.15. In order to satisfy Eqns. 4.20 and 4.21, the value of τ_r should be such that $5.7T < \tau_r < 8.8T$.

4.4.2. Competitive Dynamics Between Result Nodes

When there are several temporal modules in a network, the result node in each module receives inhibitory inputs from the result nodes of other modules. The result node of the module receiving the correct input sequence will be activated, while the result nodes of other modules will be inhibited, and will remain quiescent.

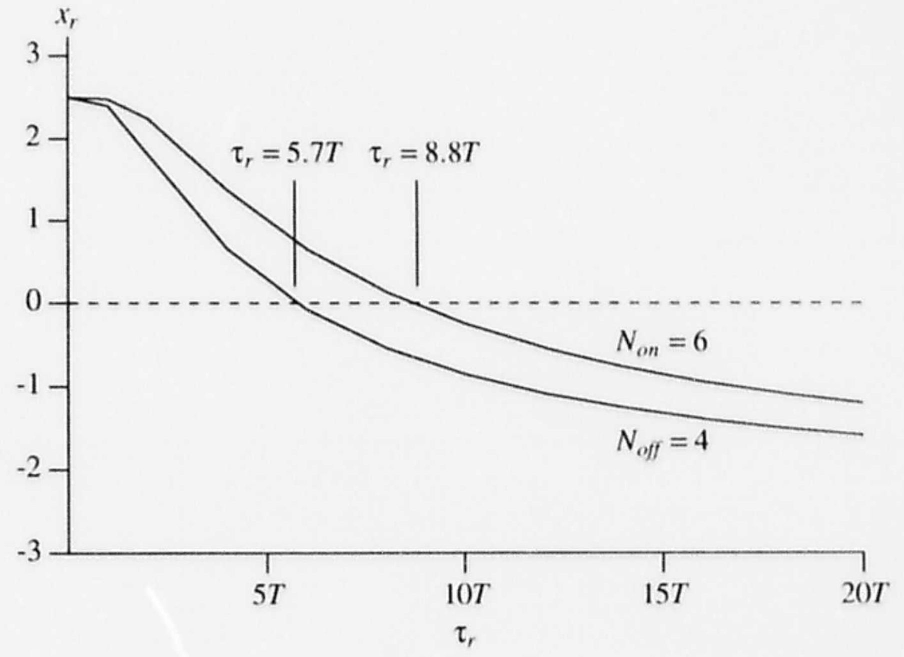


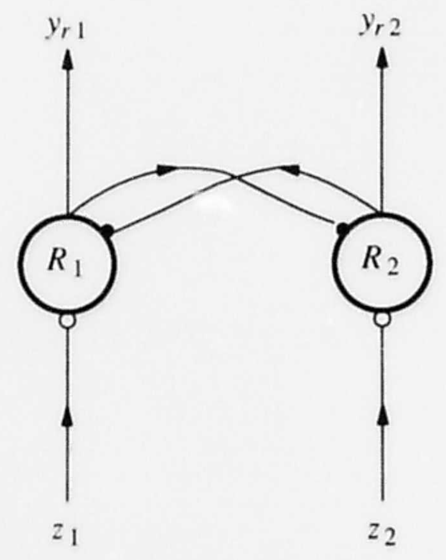
Fig. 4.15 Plot of x_r versus τ_r in terms of T for $N_{off} = 4$, $N_{on} = 6$.

4.4.2.1. Analytical Solution

To analyze the competitive process involved in the winner-take-all dynamics of multiple temporal modules, two result nodes, R_1 and R_2, are considered. They are interconnected as shown in Fig. 4.16. Each node receives an external input z_i, which represents the sum of activities coming from the corresponding sequence nodes. The dynamical equations for nodes R_1 and R_2 can be written in the form

$$\tau_r \frac{dx_{r1}}{dt} + x_{r1} = cz_1 - ay_{r2} + \theta_r \tag{4.22}$$

$$\tau_r \frac{dx_{r2}}{dt} + x_{r2} = cz_2 - ay_{r1} + \theta_r \tag{4.23}$$



○ excitatory connection

● inhibitory connection

Fig. 4.16 Two competing result nodes

where τ_r is the integration time-constant, x_{ri}, y_{ri} are the internal state and output, respectively, of node i, c is the weight of the external input link, z_i is the external input to node i, a is the weight of inhibitory connections between the nodes, and θ_r is the internal bias in each node.

Consider the following inputs to nodes R_1 and R_2 in Fig. 4.16

$$z_1 = K_1 u(t)$$

$$z_2 = K_2 u(t)$$

where K_1 and K_2 are constants and $0 \leq K_1, K_2 \leq 1$.

Solving the equations in the linear region of the piecewise linear transfer function in Fig. 4.1 yields the following responses,

$$x_{r1}(t) = \frac{[(cK_1 + \theta_r - \frac{a}{2})\beta_r(t) - \frac{a}{5}(cK_2 + \theta_r - \frac{a}{2})\alpha_r(t)]u(t)}{1-a^2} + e^{\frac{-t}{\tau_r}}(x_{r1}(0)\cosh \omega t - \frac{a}{5}x_{r2}(0)\sinh \omega t)u(t) \tag{4.24}$$

$$x_{r2}(t) = \frac{[(cK_2 + \theta_r - \frac{a}{2})\beta_r(t) - \frac{a}{5}(cK_1 + \theta_r - \frac{a}{2})\alpha_r(t)]u(t)}{1-a^2} + e^{\frac{-t}{\tau_r}}(x_{r2}(0)\cosh \omega t - \frac{a}{5}x_{r1}(0)\sinh \omega t)u(t) \tag{4.25}$$

where

$$\beta_r(t) = 1 - e^{\frac{-t}{\tau_r}}\cosh \omega t - \tau_r \omega e^{\frac{-t}{\tau_r}}\sinh \omega t \tag{4.26a}$$

$$\alpha_r(t) = 1 - e^{\frac{-t}{\tau_r}}\cosh \omega t - \frac{1}{\tau_r \omega}e^{\frac{-t}{\tau_r}}\sinh \omega t \tag{4.26b}$$

$$\omega = \frac{a}{\tau_r} \tag{4.26c}$$

and $x_{r1}(0)$ and $x_{r2}(0)$ are the initial conditions for the inputs of the nodes. Refer to Appendix C for the derivation of these solutions.

4.4.2.2. Node Responses

To examine the competitive process between two nodes, consider the case when node 1 is inactive and node 2 is active. Assume that node 1 receives an external input of 1, representing a correct input, and that the external input to node 2 is 0. This is the scenario of a result node that has been activated by a former sequence, and is being deactivated by the inhibition from another node, which is now receiving a correct sequence. The initial conditions of the nodes in the linear region of the piecewise linear transfer function are $x_{r1}(0) = -2.5$ and $x_{r2}(0) = 2.5$, and the external inputs to the nodes are $K_1 = 1$ and $K_2 = 0$.

Let us now investigate how the weight of the inhibitory connections between two nodes affects the competitive process. Substituting the initial conditions and external inputs into Eqns. 4.24 and 4.25, the responses of nodes R_1 and R_2 are as shown in solid lines in Fig. 4.17. The two dotted lines in the figure are the responses obtained from simulation using a sigmoidal transfer function at each node. The figure depicts the responses of nodes R_1 and R_2 when the weights of the inhibitory connections between the two nodes are weak ($a < 5$). The internal state x_{r1} changes from -2.5 to a value greater than 0 in a period less than τ_r. At the same time, the inhibition from node R_1 causes the internal state x_{r2} to change from 2.5 to a value less than 0. However, when the weights of the inhibitory connections between nodes are strong ($a > 5$), the competition between two nodes does not take place. The responses of the nodes in the latter case are shown in Fig. 4.18. Node R_1 cannot be activated by the external input in a period of τ_r due to the strong inhibition from node R_2. In Fig. 4.17, one should note that the results obtained by using piecewise linear analysis are close to the results obtained from simulation using a sigmoidal transfer function at each node. However, the simulation and analytical results shown in Fig. 4.18 are close to each other only for the time between 0 and $0.25\tau_r$ because the nodes no longer operate in the linear region for strong inhibitions between nodes.

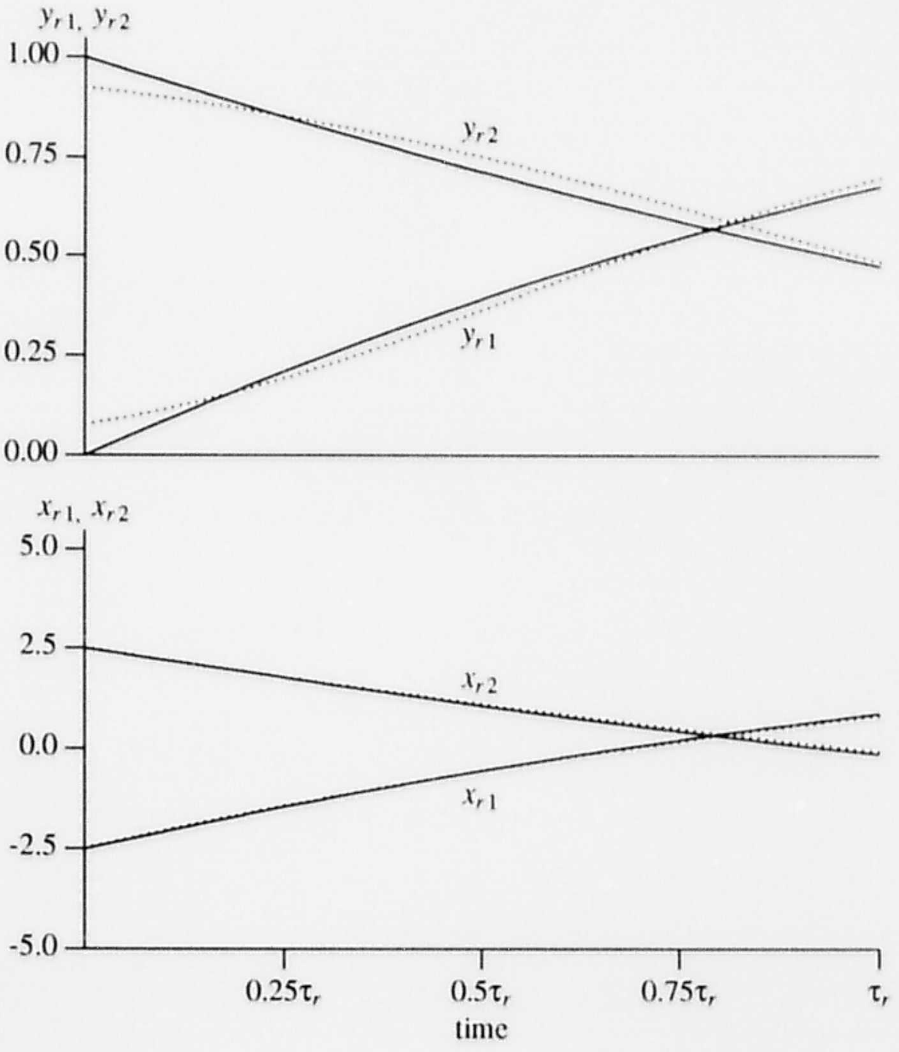


Fig. 4.17 Responses of R_1 and R_2. Solid lines are the results obtained from analytical solutions, and dashed lines from simulation.

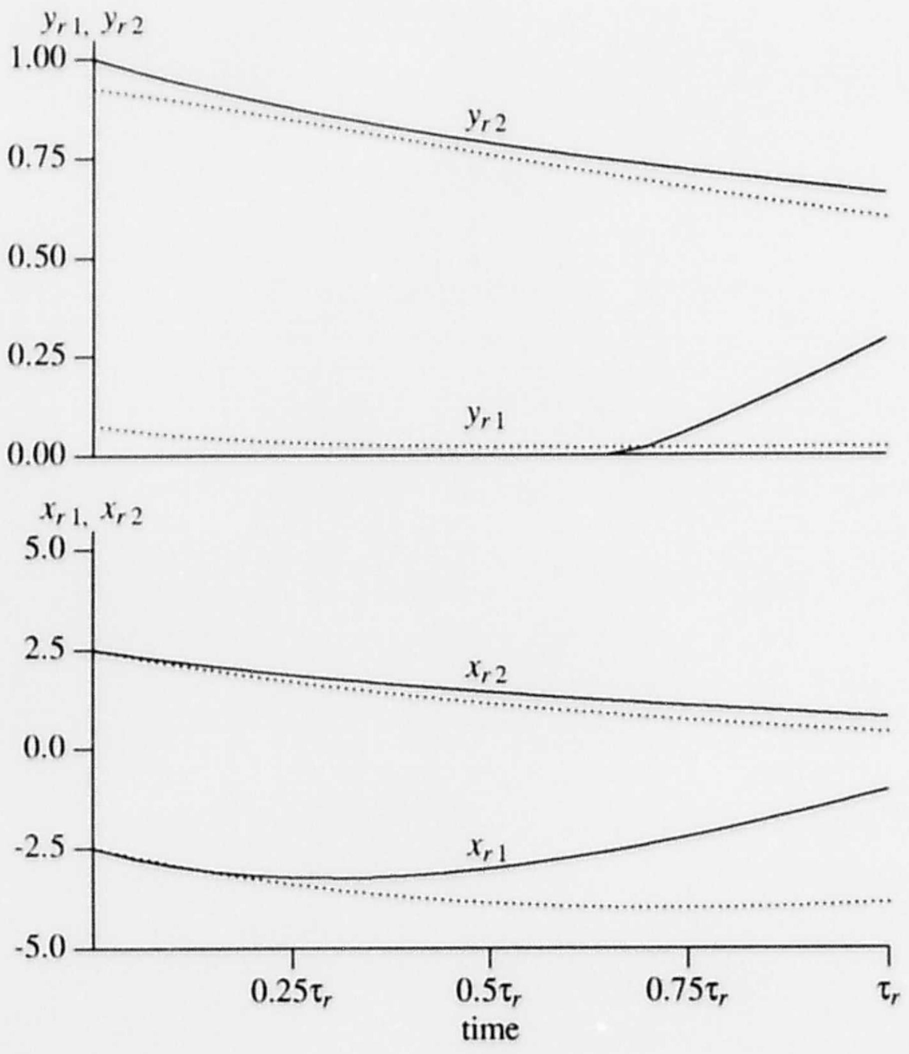


Fig. 4.18 Responses of R_1 and R_2 when a is too strong.

4.4.2.3. Estimation of Mutual Inhibitory Weight

A suitable range for a may be derived from the analytical expressions. For node R_1 to become a winner within a certain period, the internal state has to change from -2.5 to any value above 0, and x_{r2} has to change from 2.5 to any value below 0. Since the time constant τ_r is associated with how fast a node can be activated, we choose the period required for the above activities to occur to be τ_r. Substituting the values of the above initial conditions and external inputs into

Eqns. 4.24 and 4.25, the necessary conditions for a winner-take-all process to occur can be expressed as follows:

$$0 \le \frac{(c+\theta_r - \frac{a}{2})\beta_r(\tau_r) - \frac{a}{5}(\theta_r - \frac{a}{2})\alpha_r(\tau_r)}{1-a^2} - 2.5e^{-1}(\cosh a - \frac{a}{5}\sinh a) \tag{4.27}$$

$$0 \ge \frac{(\theta_r - \frac{a}{2})\beta_r(\tau_r) - \frac{a}{5}(c+\theta_r - \frac{a}{2})\alpha_r(\tau_r)}{1-a^2} + 2.5e^{-1}(\cosh a + \frac{a}{5}\sinh a) \tag{4.28}$$

where β_r and α_r are expressions given in Eqns. 4.26a and 4.26b, respectively.

Given the values of c and θ_r, the range of a for which the nodes will operate in a winner-take-all fashion can be estimated using the above conditions. However, it is only an approximation, because the equations do not account for the non-linearity in the transfer function of a node. In the case of a multi-module system, the value of a derived above provides an upper bound. The total inhibition from several partially triggered nodes in a multi-module system may be greater than the inhibition from one fully active node in a two-module system. The strong inhibition from multiple modules may prevent the firing of the result node in a module receiving a correct sequence. Hence, to ensure proper activation of a node receiving inhibitions from other partially triggered nodes, the inhibitory weight a should not be greater than the value given by the upper bound defined by the analysis for a two-node system.

To demonstrate the application of the two conditions in Eqns. 4.27 and 4.28, consider a network with the following parameters: $\tau_r = 10^{-3}$, $c = 4$ and $\theta_r = -0.4$. By substituting the values of the parameters into Eqns. 4.27 and 4.28, we obtain $0 \le a \le 7.5$. Let us now investigate the accuracy of the estimated range for a by comparing it with the range of a obtained by using a sigmoidal transfer function at each node in a simulation. The initial conditions used were $x_{r1}(0) = -2.5$ and

$x_{r2}(0) = 2.5$, and the external inputs were $K_1 = 1$ and $K_2 = 0$. The solid and dotted lines in Fig. 4.19 are a plot of the time for $x_{r1}(t)$ to rise from -2.5 to a value just above 0 and to decay from 2.5 to a value just below 0, respectively, as the weight a varies. For node R_1 to be activated and node R_2 to be deactivated after a period τ_r, the weights should satisfy $0.38 \leq a \leq 3.75$. Therefore, the range of a obtained from the analytical solution is an acceptable estimate for the mutual inhibitory weight, a, between modules.

4.4.3. Summary

We have analyzed in this section how a result node in a module can be used to integrate in time the activities of the sequence nodes in the same module to generate an output representing a

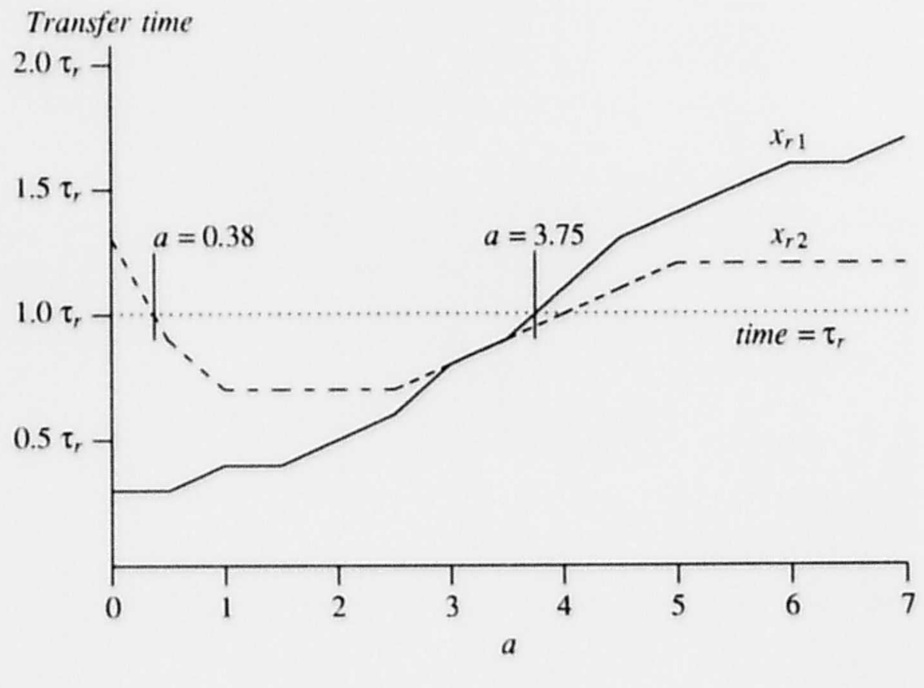


Fig. 4.19 Plot of the required time for internal states x_{r1} and x_{r2} to fall from 2.5 to a value less than 0 and to rise from -2.5 to a value greater than 0, respectively, versus a.

temporal sequence, and the competition between the result nodes of two modules where one becomes a winner. The analysis has also provided a way to estimate the parameters such as the integration time constant of the result node for a given application. The estimation of the network parameters is used in the simplified training algorithm, which will be presented in the next section.

4.5. Simplified Training Algorithm

The training algorithm [illegible]ented in chapter 3 requires a long computational time to train a temporal module, because weights are adjusted in small increments.

This section presents a simplified algorithm to train a temporal module, in which only the weights of the input interconnects are adjusted. The weights of the interconnects between nodes are computed using the analytical results presented in the earlier sections. The training algorithm consists of the following steps.

(1) For a given application, where the sampling period T is known, determine the number of sequence nodes needed using $N = \frac{T_{total}}{T}$ where T_{total} is the duration of interest.

(2) Choose the time constant of the sequence nodes to be $T \leq \tau_s \leq 2T$ and the time constant of the result node to be $\tau_r = n \times T$ where $n \leq N$.

(3) Determine suitable interconnect weights between sequence nodes to the node to be in the middle of the range defined by the inequalities 4.14 and 4.15. Determine the input connect weight to the result node using 4.20 and 4.21. Finally, determine the mutual inhibitory weights between modules using 4.27 and 4.28.

(4) Uncouple all sequence nodes in a module, and train each node to be partially activated by the associated event in a training sequence. To speed up the training process of each sequence node, compute the starting values for the input connect weights using Eqn. 4.17.

(5) After all the sequence nodes have been trained, link the sequence and result nodes in a module together using the interconnect weights from step 3. These weights may require

some fine-tuning, because of the approximations used in the analytical model.

(6) Repeat steps 2 and 3 for each module in the network.

(7) After each module is trained to recognize a temporal sequence, interconnect all the modules in a network using the interconnect weights from step 3. The internal bias of the result node in each module is then fine tuned by presenting the complete training sequence to the network.

The above algorithm provides a simple and modular approach to the training of the proposed network. It has been used to train larger networks than the one presented in chapter 3. Simulation results of several large networks trained by this algorithm are presented in Chapters 5 and 6.

4.6. Conclusion

This chapter has used a piecewise linear model to analyze the behavior of the proposed temporal module. The analysis yields an estimate for parameters such as the sampling period and the interconnect weights. The temporal integration process in a result node was also analyzed. The analysis of the winner-take-all dynamics between multiple result nodes for neural networks with many temporal modules provides an understanding of the inhibitory process between the nodes when they compete with each other in a winner-take-all fashion.

Chapter 5

Application in Motion Detection

5.1. Introduction

Motion detection in computer vision is an important task. Image input is sampled at regular intervals and presented to a detector as a sequence of images, one image at a time. A massive computational effort is required because a motion detector needs to compare successive images in a sequence, taking into account the elapsed time between them. After examining a sufficient number of samples, it should decide whether the sequence represents a moving object. Several mathematical models, software algorithms and experimental systems have been proposed either to achieve motion detection in real time or to understand the computational process involved in a biological vision system [for example, Barlow 1964, Barlow 1965, Curlander 1987, Poggio 1987, Sivilotti 1987, Hutchingson 1988, Mead 1988, Yuille 1988, Mead 1989].

One approach to meet the large computational requirement of a motion detector is to utilize the parallel processing capability of a neural network. This chapter presents the implementation of a motion detector using the temporal modules presented earlier. A number of such modules are used, where each module is trained to recognize one particular motion of an object along a given trajectory. Once all the modules have been trained, they can be used to detect the movement of an object.

The purpose of this chapter is to illustrate the application of the modules proposed in chapter 3 to motion detection by providing realistic test inputs. It is not meant to be a detailed study on motion detection.

The chapter is divided into three sections. It begins by introducing some of the existing work on motion detection, and then describes the architecture of the proposed motion detector. A performance study of a motion detector based on one module with a few sequence nodes is presented, followed by some simulation results of larger networks.

5.2. Existing Approaches to Motion Detection

Several architectures that are not based on the neural network approach have been proposed for real-time motion detection [Dickmanns 1988a, Dickmanns 1988b, Dickmanns 1989, Burt 1989, Waxman 1989]. For example, the architecture proposed by Dickmanns uses a fast multi-processor system to achieve real-time motion detection [Dickmanns 1988a, Dickmanns 1988b, Dickmanns 1989]. This system is based on the Kalman filter approach to recursive state estimation to detect the motion of rigid objects when a sequence of images is presented. The feature position and angular orientation of each object relative to the camera are computed for every image presented and are used for estimating the state variables of the Kalman filter based on the differences between the predicted and the measured feature positions. A direct spatial interpretation including velocity components can be derived from the prediction errors. This architecture has been used to guide a test vehicle moving at high speeds on a highway under various weather conditions.

When an object moves, the intensity at some pixels of the image changes. A popular algorithm to detect motion is to compute the spatial and temporal intensity change at every pixel of an image [Longuet-Higgins 1980, Schunck 1984]. The result of this computation is a pattern of velocity vectors called optical flow, which provides the location and motion of the object. To compute the optical flow pattern at each successive image requires intensive computation. Thus real-time computation of optical flow cannot be achieved on current sequential computers. One approach to meet the real-time requirement of computing optical flow is to exploit the parallel computation ability of a neural network on a VLSI chip. Some VLSI chips that do real-time computation of optical flow have been implemented [Hutchinson 1988, Koch 1989, Mead 1989].

Other forms of neural networks that compute visual motion have also been proposed. For example, a neural network to account for short range motion detection in the insect visual system was proposed by Ogmen and Gagne [Ogmen 1990]. The network is based on a model in which two input signals from two nodes at successive time steps may interact in product form using a "gated dipole", in addition to the spatial summation of weighted incoming signals at each node. This interaction simulates the pre-synaptic inhibition that gives a neuron the ability to detect motion in a biological vision system. Sereno used a multi-layer feed-forward neural network, together with a layer of local detectors capable of detecting the component of motion perpendicular to the orientation of the moving edge, to recognize the global movement of randomly oriented contours [Sereno 1989]. In another neural network proposed by Marshall, a sequential neural network was used to perform motion detection [Marshall 1990]. The network is trained in such a way that only nodes located along the direction of motion are linked by excitatory connections. Otherwise they are linked by inhibitory connections. As a result, the nodes linked by excitatory connections are activated accordingly when an object moves along the training path. Any other direction of motion causes the nodes to be inhibited.

5.3. Proposed Motion Detector

Given a sequence of images of an object moving in a plane parallel to the image plane, the function of the proposed motion detector is to estimate quantities such as the speed and direction of the movement of an object. The detector does not perform any spatial pattern recognition function. It recognizes the motion of any object as long as the object moves along the trajectory that the detector has been trained to recognize and has a similar physical size to the size of the training pattern.

Image input to the detector consists of a two-dimensional array of binary pixels. A sequence of images is generated and presented to a detector by sampling the input at regular intervals. The motion detector is organized in three layers: input sampling, summing, and sequence recognition, as shown in Fig. 5.1.

Motion detection is accomplished by comparing each two successive images in a sequence to determine the difference between the current and previous images. An object is regarded as moving when changes in its position are detected in two or more successive images. Motion is detected by providing for each pixel in the image input a discrete time differentiator that generates an output of one if the present input is different from its previous value, and zero otherwise. A moving edge

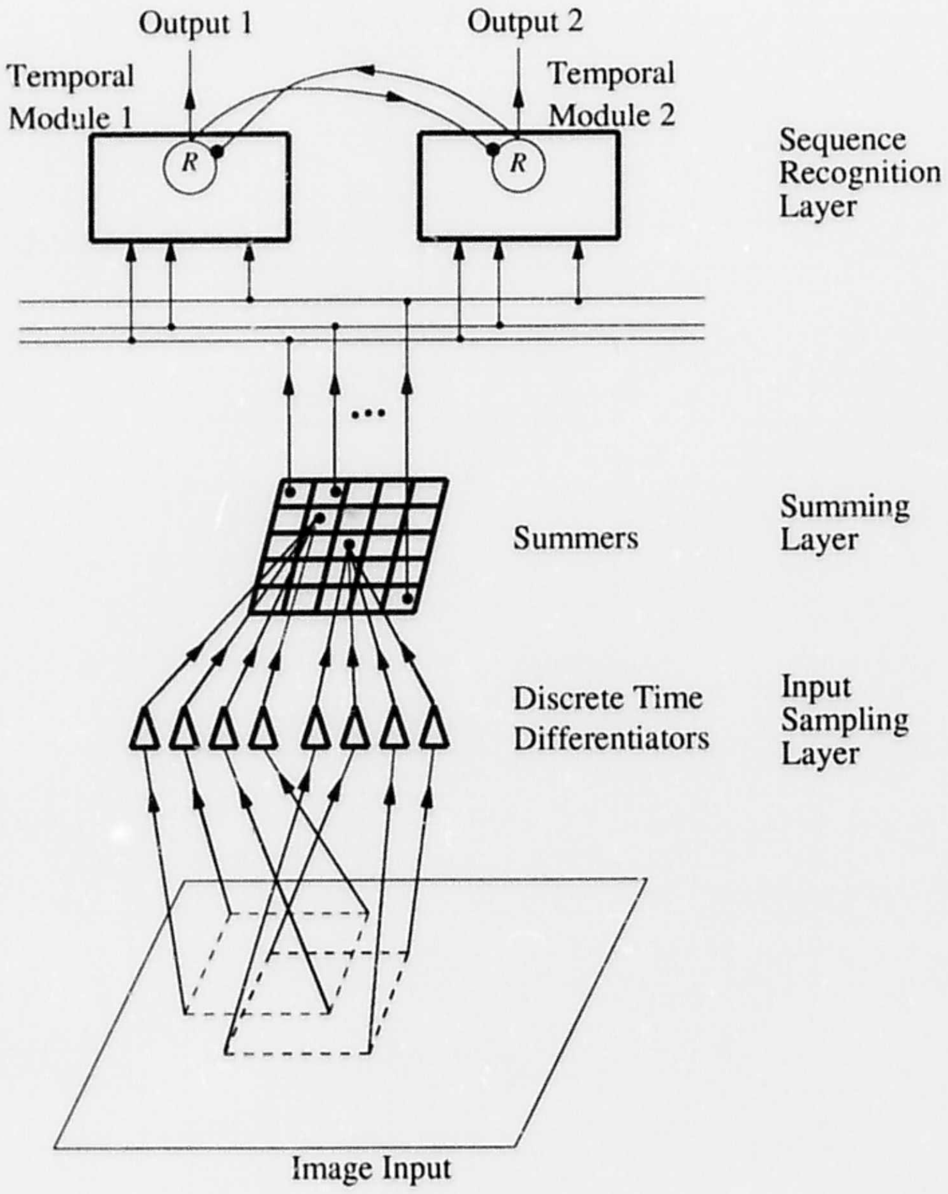


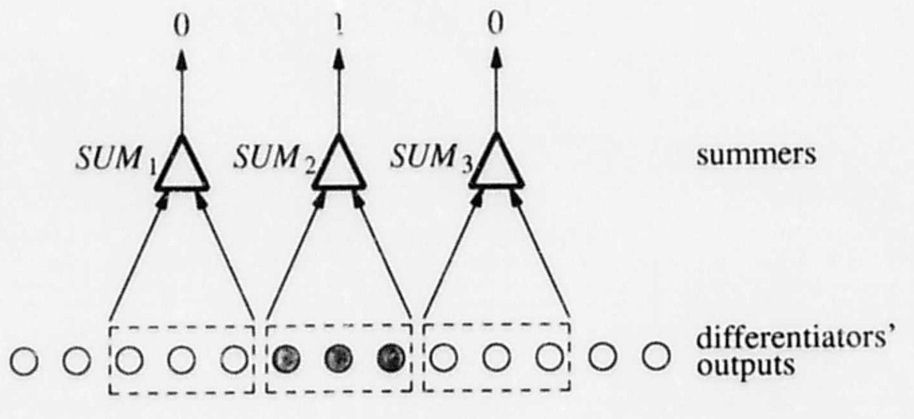
Fig. 5.1 Architecture of a motion detector

causes a change in brightness at some pixels, hence causes non-zero temporal derivatives to be generated by the corresponding differentiators.

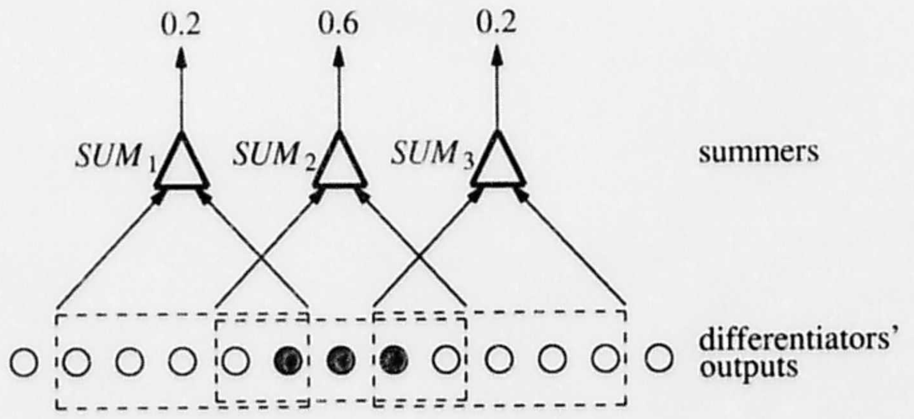
Connecting the output of each differentiator directly to the sequence recognition layer is infeasible, because the image input comprises a large number of pixels requiring many interconnections. To reduce the number of interconnections needed, a summing layer is used. It consists of a two-dimensional array of linear summers. The image is divided into regions called windows, and each summer is associated with the pixels in a window. A summer sums the magnitude of the time derivative of the signals from each pixel in the associated window at each sampling period, and gives an estimate of the activity occurring in the window. The summer's output will be between 0 and 1, depending on the density of pixels that have changed in brightness during a sampling period.

The windows associated with the summers may overlap, as shown in Fig. 5.1 and in more detail in Fig. 5.2. This overlapping of windows has been used by others in certain architectures for pattern recognition [Rumelhart 1986, Fukujima 1988]. It acts as a lowpass filter on the summers' output. The inputs of the summers are connected to the outputs of the differentiators, which are represented as circles in the figures. For a differentiator with an output of 1 a solid circle is used, and for an output of 0 an open circle is used. If the outputs of all the differentiators in a window associated with a summer are 1, then the output of that summer will be 1. Otherwise, the summer's output will be a fraction of 1 depending on how many differentiators in the window are turned on. The windows in Fig. 5.2a do not overlap. For the output pattern shown in the figure, summer SUM_2 has an output of 1, whereas the other summers have outputs of 0. When 40 % window overlap is used, as shown in Fig. 5.2b, the same output pattern of the differentiators causes summer SUM_2 to have an output of 0.6, and summers SUM_1 and SUM_3 to have outputs of 0.2 each. With more overlap, the summer's output changes less abruptly as the object moves.

The sequence recognition layer consists of several temporal modules, each trained to recognize a specific trajectory of an object in time and space. All modules receive the same input from the summing layer, and each of the modules keeps track of the difference between two successive images caused by a moving object. When there is more than one temporal module in the sequence



a) no window overlap



b) 40 % window overlap

Fig. 5.2 An array of summers with their respective windows.

recognition layer, the movement of an object along a trajectory that is close to one of the training trajectories will cause one of the modules to be activated. Ideally, the other modules should remain quiescent. However, they can sometimes be partially activated due to noise or to interference from the trajectory presented. To prevent the partial activation of the modules, they are operated in a winner-take-all fashion, with each module receiving inhibitory inputs from all other modules. The module recognizing the correct motion sequence achieves the maximum output while outputs of all the other modules are inhibited.

5.4. Estimation of Weights

Suppose we want to design a simple motion detector that recognizes the rightward motion of a w-pixel solid object at a speed of w pixels per unit time against a white background. The object motion is sampled and held at every 10^{-4} seconds, generating a sequence of 11 images.

Consider a motion detector shown in Fig. 5.3. The temporal module consists of ten sequence nodes and one result node. The window associated with each summer has a size of w pixels. Summer i will produce an output of $\frac{x}{w}$ whenever x pixels in window W_i change their value in a unit time. To enable sequence node S_i to detect the motion of a w-pixel object from window i to $i+1$, the node inputs are connected to the outputs of summers SUM_i and SUM_{i+1} by excitatory connections and to other summers by inhibitory connections. By interconnecting the sequence nodes in a chain, as described in chapter 3, it is possible to have a detector that can recognize the motion of an object specified above.

Since the sampling period T is given, the system parameters can be estimated using the analytical results from sections 4.3 and 4.4 of chapter 4, as shown in Table 5.1. The motion detector was simulated using these parameters. Output y_R of the result node to the rightward and leftward movement of a w-pixel object at a speed of w pixels every T seconds is shown in Fig. 5.4. The figure shows that the detector is able to distinguish between the two types of motion, even though the output is a bit low. The low output is due to approximations made in chapter 4, where it was assumed that all the nodes are operating in the linear region of the sigmoidal output function. However, the result indicates that the analysis given in chapter 4 is sufficient to provide an estimate of the parameters.

Using the parameters in Table 5.1 as the initial values for the interconnect weights between nodes, the network is then trained to recognize the two motion sequences using the LMS algorithm given in chapter 3. Fig. 5.5 shows the responses of the trained detector to an object moving to the

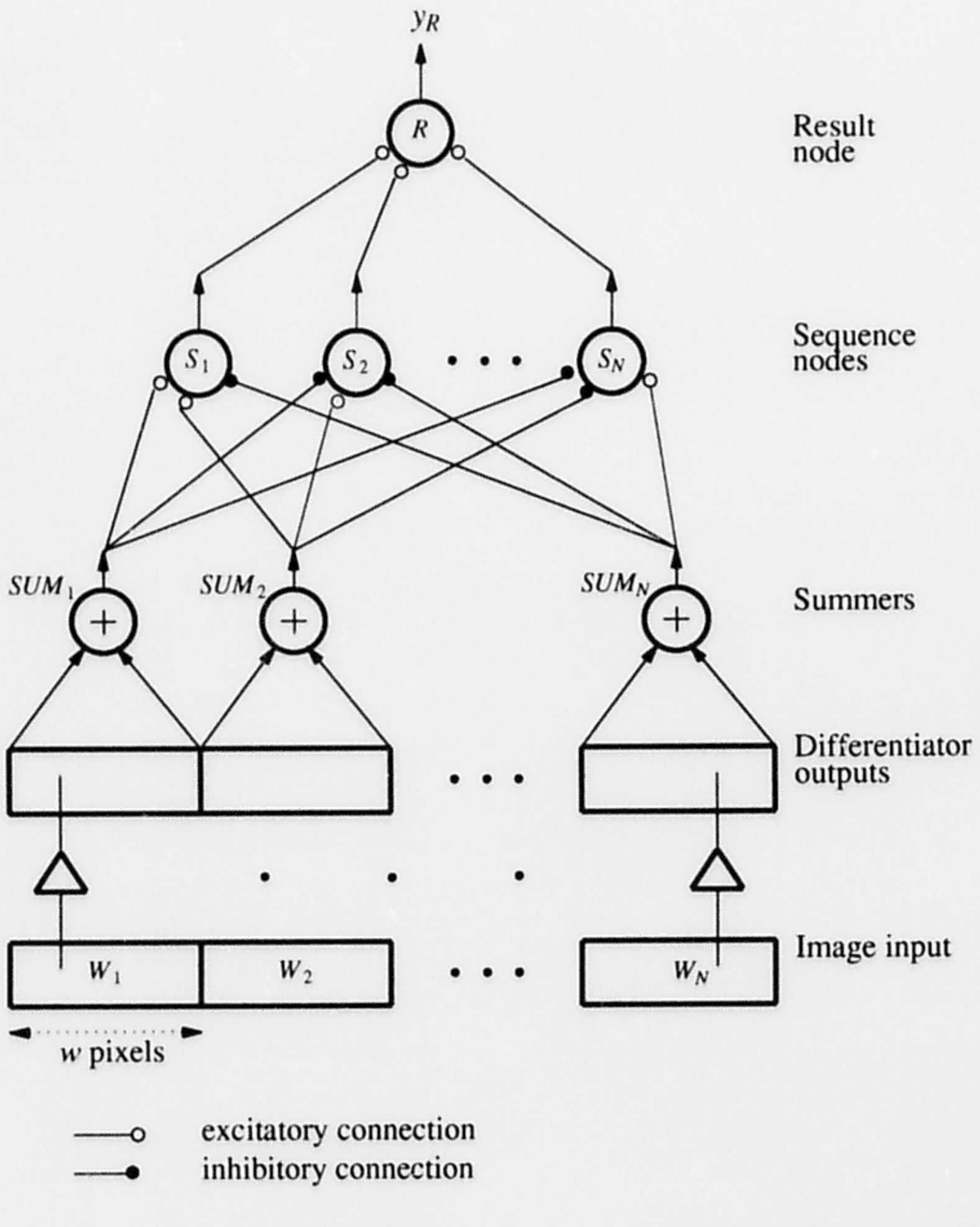


Fig. 5.3 A motion detector to detect a linear motion.

parameters	values
τ_s	10^{-4}
θ_s	-0.1
a	1.5
b	0.5
τ_r	5×10^{-4}
θ_r	-0.1
c	1.0

Table 5.1 Estimated system parameters obtained from analytical results.

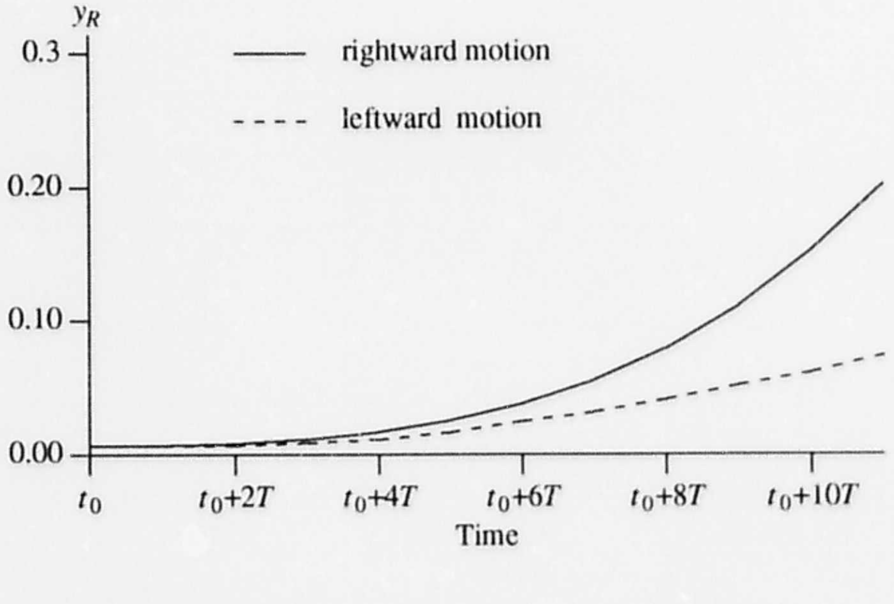


Fig. 5.4 Output of result node when an object moves to the right and left using weights derived from analysis.

right or left. Unlike the responses in Fig. 5.4, the output of the detector is close to 1 when the object is moving from left to right, and is below 0.2 otherwise. The results show a much better ability to distinguish between leftward and rightward motion.

Table 5.2 gives the outputs of the sequence nodes for each period while the object is moving from left to right. This table is provided so that comparison can be made when a different input condition is introduced. During the first period, node S_1 is excited by summers SUM_1 and SUM_2, and thus its output rises from 0.0 to 0.4 at time t_0+2T. The output of node S_2 rises to 0.5 at t_0+2T, then to 0.8 at time t_0+3T. The higher excitation values at node S_2 are due to the fact that the node has been pre-excited by node S_1. The output of subsequent nodes increases in a similar manner. Therefore, when the excitation is received by the nodes at the right time, they are strongly activated from left to right, one node at a time. The time integral of the sequence node outputs computed by the result node gives rise to the response shown by solid line in Fig. 5.5. The output of the result

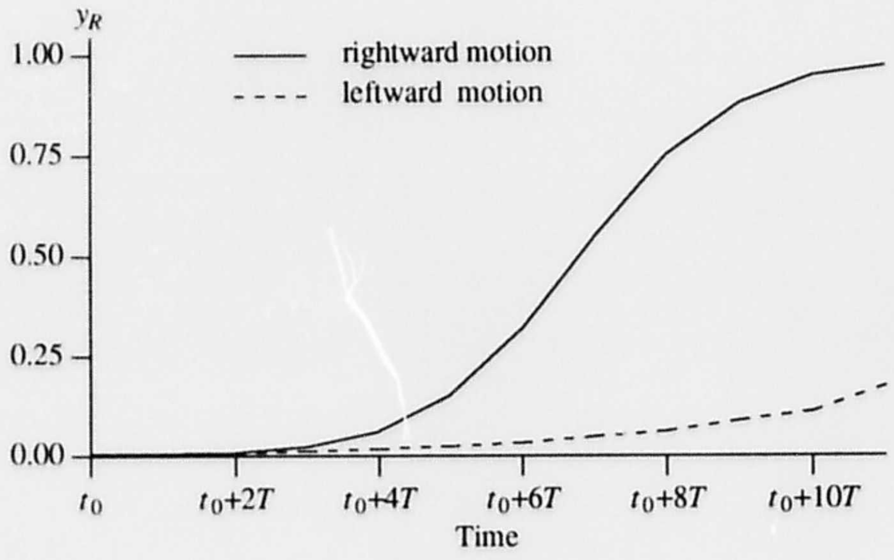


Fig. 5.5 Output of result node when an object moves to the right and left after training.

node at time t_0+11T is 0.977.

5.5. Training Rate

In this section, we want to determine if there is any speed-up in the training rate of a temporal module when initial weights are pre-estimated using the analysis given in chapter 4, as compared with that of a module having zero initial weights. The comparison is achieved by measuring the number of iterations required to train a module such that the training error is lower than a certain threshold.

Two types of training error can be measured when a temporal module is trained to recognize a sequence of images representing the movement of an object. One is the error generated by the sequence nodes and the other one is the error generated by the result node. The training error generated by a chain of sequence nodes in one training iteration was measured as follows. First, the sum of the squares of the errors at all nodes in the chain is obtained for each event presented. Then,

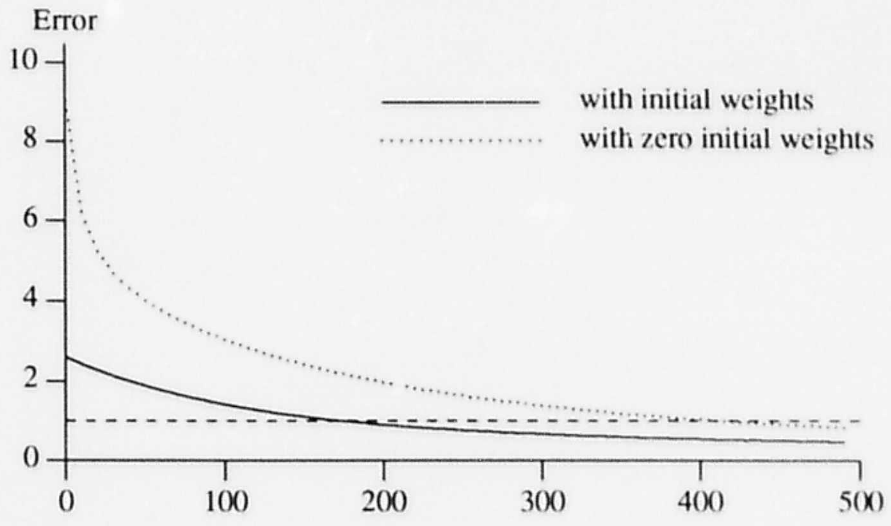
time	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
t_0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
t_0+T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
t_0+2T	0.4	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
t_0+3T	0.0	0.8	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
t_0+4T	0.0	0.0	0.9	0.5	0.0	0.0	0.0	0.0	0.0	0.0
t_0+5T	0.0	0.0	0.0	0.9	0.5	0.0	0.0	0.0	0.0	0.0
t_0+6T	0.0	0.0	0.0	0.0	0.9	0.5	0.0	0.0	0.0	0.0
t_0+7T	0.0	0.0	0.0	0.0	0.0	0.9	0.5	0.0	0.0	0.0
t_0+8T	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.5	0.0	0.0
t_0+9T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.5	0.0
t_0+10T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.5
t_0+11T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9

Table 5.2 Outputs of sequence nodes at each period when an object of w pixels moves to the right at a speed of w pixels per unit time.

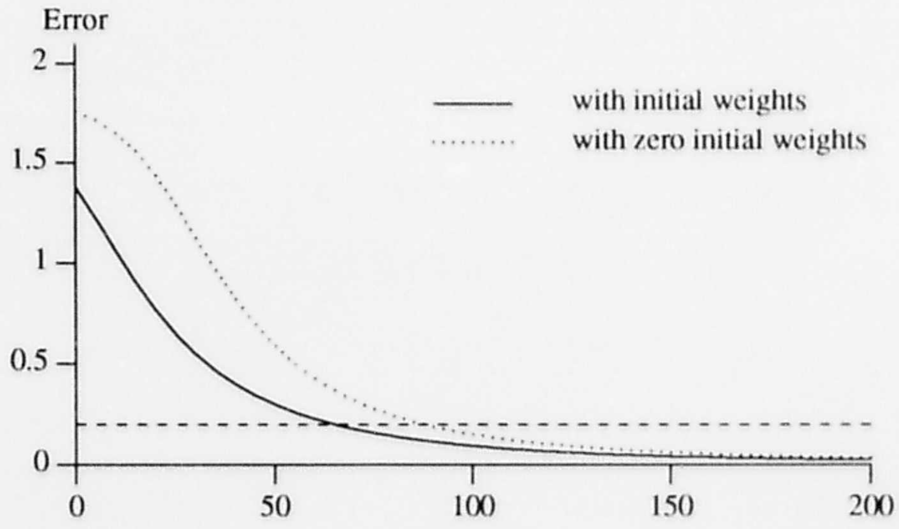
the sum of the errors for all events in the training sequence is obtained. One training iteration is completed at the end of each sequence. After the sequence nodes have been trained, the result node in the same module is then trained to generate a correct response at the right time. To measure the error generated by a result node, the square of the error at the result node is computed for each event presented. A sum is obtained by adding the square of error at the result node for every event in a sequence. The end of a sequence completes a training iteration.

A typical training curve for one temporal module of a motion detector is shown in Fig. 5.6. During training, there are two criteria to determine whether a temporal module has been sufficiently trained to recognize a motion sequence. The first relates to the absolute error produced by a module, and the second to the rate of convergence which is defined as the change in absolute error in one training iteration. A threshold value is established for each of these errors, and the training of a module is terminated if either error is less than the corresponding threshold. The threshold value for absolute error was chosen to be approximately 10% of the maximum training error when zero initial weights are used. The training curves given in Fig. 5.6a show that the threshold of 1.0 was reached after approximately 200 iterations when the initial values of all interconnect weights obtained by analysis were used. Approximately 400 iterations were needed otherwise. This amounts to a 50% saving in training time. The rate of convergence in both cases was greater than the threshold of 0.001.

Fig. 5.6b shows that a threshold of 0.2 was reached after the result node had been trained for approximately 60 iterations, using the initial weights obtained by analysis. Again, the node takes about 1.5 times as long to train when the zero initial weights are used. The result node requires a lesser number of training sessions than the sequence nodes because there are fewer interconnections that need to be adjusted.



a) Sequence nodes.



b) Result node

Fig. 5.6 Absolute training errors of sequence and result nodes. The solid line shows the training error when initial interconnect weights are provided. The dotted line otherwise.

5.6. Performance Study

The response of the motion detector in Fig. 5.3 to objects of different sizes moving at different speeds is examined in this section. Comparison of the movement of a solid object against a white background and the movement of a random-dot object against a random-dot background is also presented.

5.6.1. Object Size

Consider now the case of an object whose size is different from the training size of w pixels. The node outputs for object sizes of $0.4w$, w, and $1.8w$ pixels are shown in Fig. 5.7. The figure shows that when the object size is different from the training size, the sequence nodes have lower outputs than the case when the size is w pixels. Naturally, this leads to a decreased output at the result node.

The lower sequence node outputs can be explained as follows. When the size is less than w pixels, fewer pixel changes are detected by the corresponding summer of a sequence node. The outputs of summers SUM_1, SUM_2, SUM_3 and SUM_4 are shown as dashed lines in Fig. 5.8 when the object size is $0.4w$ pixels.

An object of size w pixels causes only two neighboring summers to be activated during any interval T. When the object size is greater than w pixels, other neighboring summers are also activated. For example, summers SUM_1 and SUM_2 are activated between the time t_0+T and t_0+$2T$ when the object size is w pixels, while summers SUM_1, SUM_2 and SUM_3 are activated during the same period when the object size is $1.8w$ pixels. Since neighboring summers are connected to the inputs of a sequence node by inhibitory connections, the node's output will be reduced.

Fig. 5.9 presents the output of the result node as the object size is varied. We shall take the level L_{on} = 0.5 as the level at or above which the module is regarded as having recognized the motion of the object. The figure shows that the module can detect the movement of an object with a size between $0.5w$ and $1.6w$ pixels.

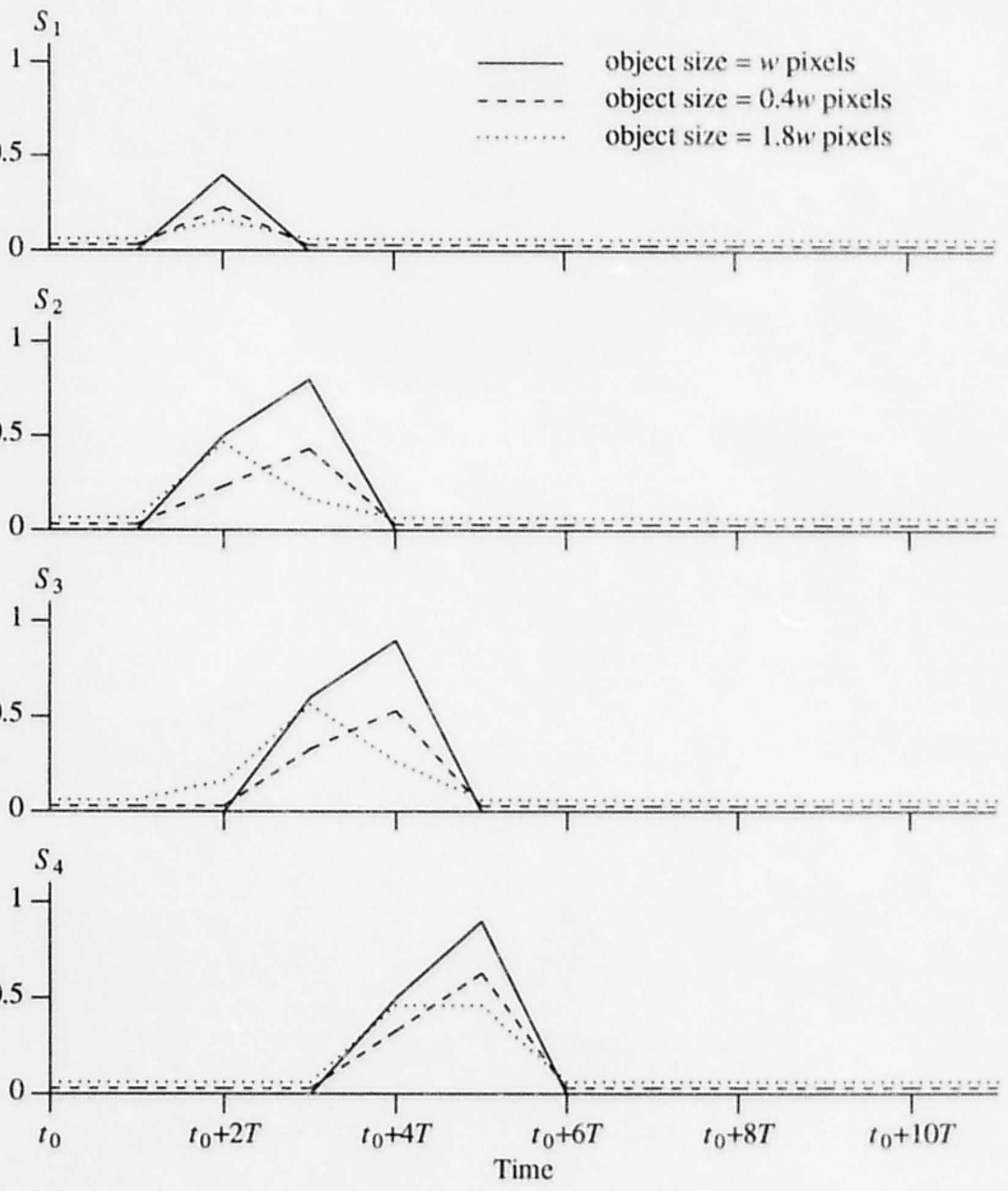


Fig. 5.7 Outputs of sequence nodes when objects with the sizes of $0.4w$, w, and $1.8w$ pixels move to the right at a speed of w pixels per unit time.

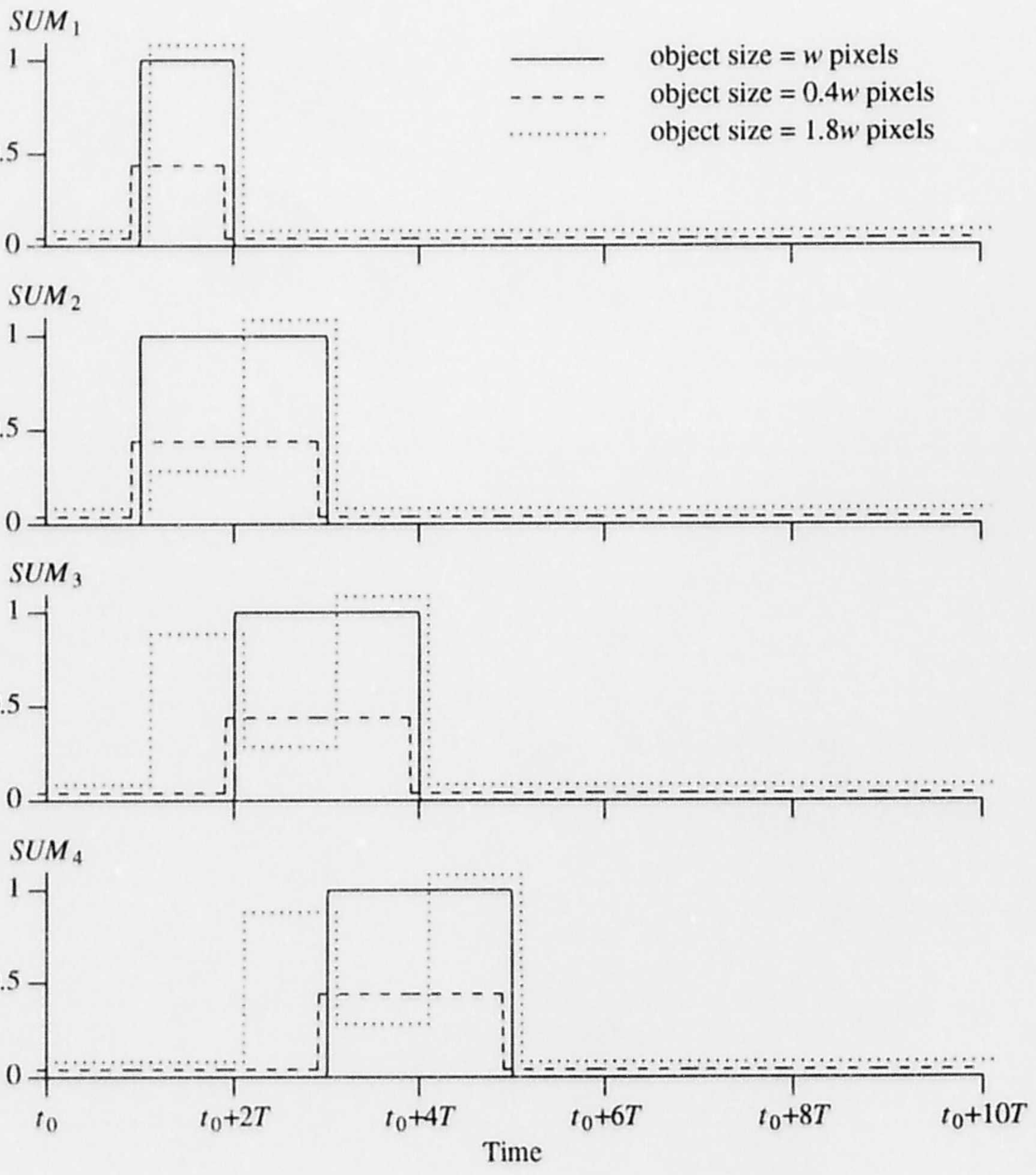


Fig. 5.8 Outputs of summers when objects with the sizes of $0.4w$, w, and $1.8w$ pixels move to the right at a speed of w pixels per unit time.

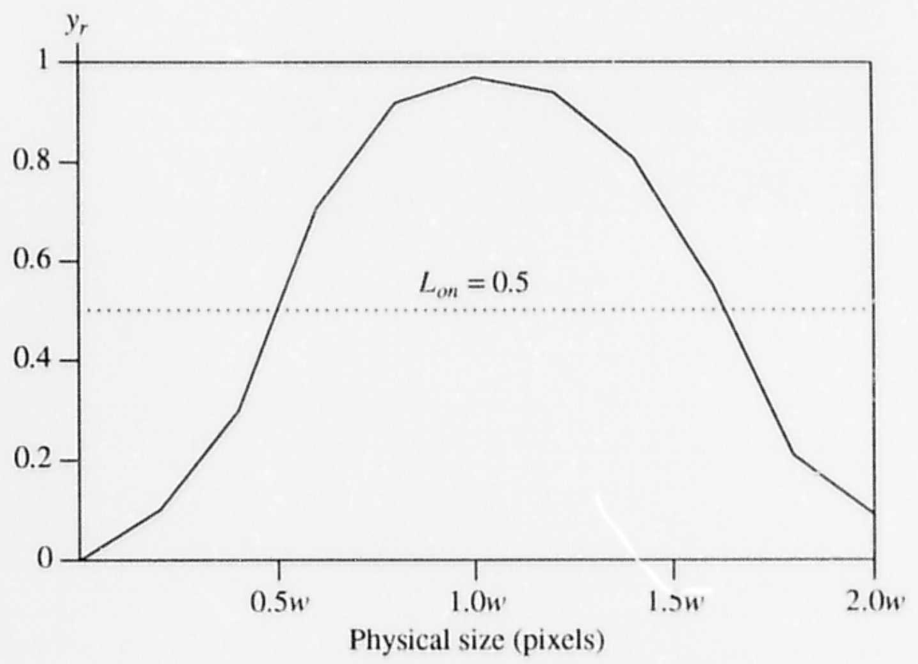


Fig. 5.9 Output of the motion detector at time (t_0 + 11T) versus the object size.

5.6.2. Object Speed

The response of the motion detector to the movement of an object at different speeds is examined in this section. The size of the object used in this part of the study is w pixels. Fig. 5.10 shows the node outputs when the speed is less than and greater than the training speed of w pixels per unit time.

When the speed is less than w pixels per unit time, each sequence node is partially activated during successive sampling periods, because the object takes a longer time to move across the window associated with that node. This is illustrated by the summer outputs given in Fig. 5.11. For example, when the motion speed is w pixels per unit time, summer SUM_4 is fully activated between the time t_0+3T and t_0+5T. However, when the motion speed is 0.6w pixels per unit time, the summer is only partially activated, and the period of activation is between t_0+4T and t_0+8T.

The lower summer outputs and activation of the sequence nodes at the wrong time cause the sequence nodes to be partially activated. Thus, temporal integration of the lower node outputs by the result node will produce only a partial response.

When the object speed is greater than w pixels per unit time, the sequence nodes are also weakly activated. Let us consider sequence node S_i. When an object moves at a speed greater than w pixels per unit time, neighboring summers other than SUM_i and SUM_{i+1} are strongly activated, as shown in Fig. 5.11. For example, summers SUM_2 and SUM_3 are activated between the time t_0+2T and t_0+3T when the speed is w pixels per unit timer. When the speed is increased to $1.4w$ pixels per unit time, summers SUM_2 and SUM_3 are partially activated, along with the downstream SUM_4. Sequence node S_2 receives inhibition from summer SUM_4, and also nodes S_4 and S_5 downstream, which are activated by summer SUM_4. As a result, sequence node S_2 is only weakly activated at time t_0+3T. Summing in time the weak outputs of the sequence nodes generates a response that is lower than the case when the motion speed is w pixels per unit time.

Fig. 5.12 shows that the module can detect the movement of an object moving at a speed between $0.5w$ and $1.2w$ pixels per unit time.

5.6.3. Random-dot Image Input

In what follows, we will examine the performance of the motion detector when the test object consists of a random-dot pattern moving against a background containing a different random-dot pattern. This type of image input was chosen because it provides an easy input interface with the motion detector. It also approximates a textured planar surface moving smoothly against a planar background [Regan 1984, Adelson 1986, Egelhaal 1988]. An example of a test sequence consisting of 4 events is shown in Fig. 5.13.

In order to understand how well a motion detector will perform when random-dot images are presented, we need to examine the difference between the movement of a solid object against a white backgound and the movement of a random-dot object against a random-dot background. The

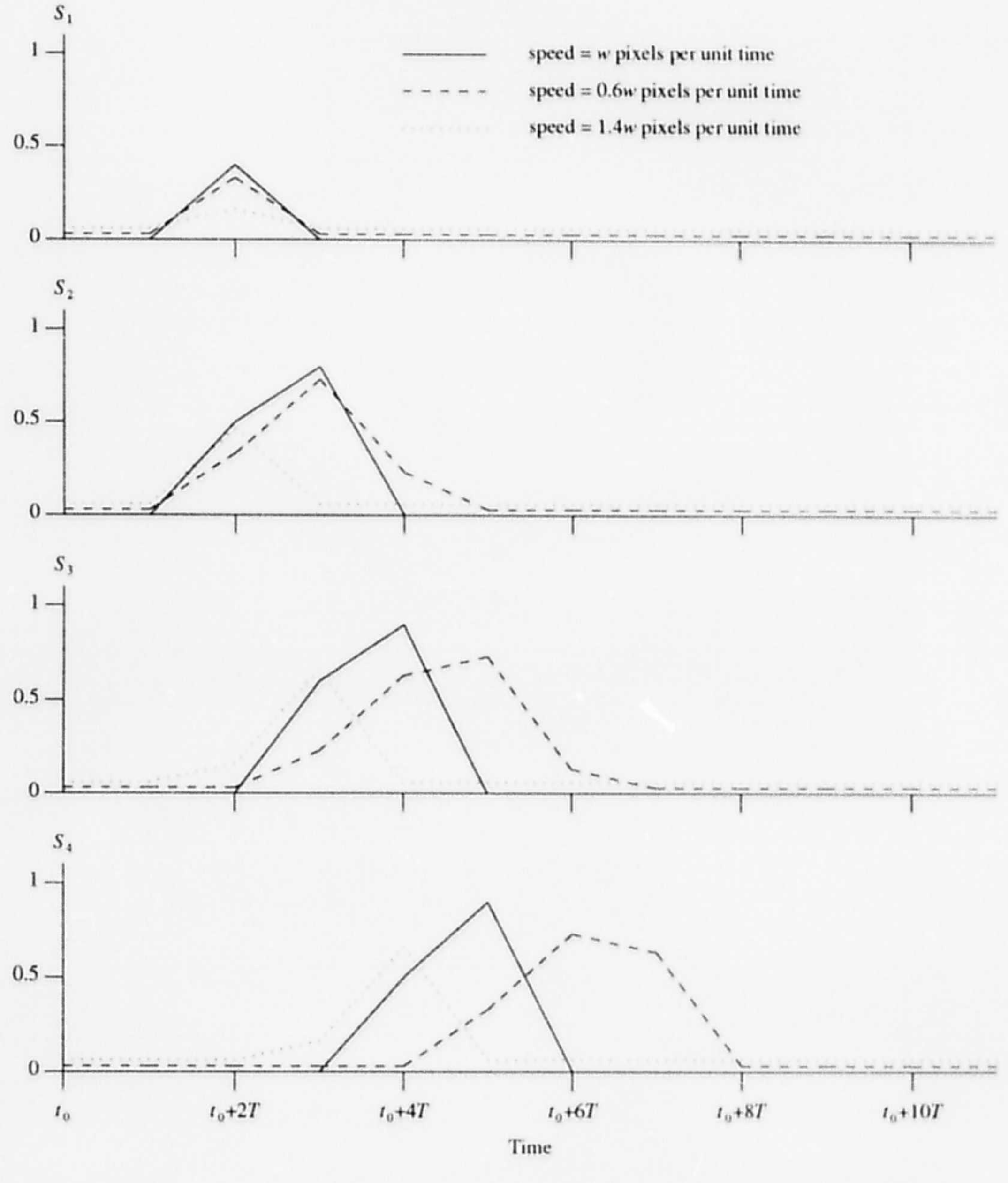


Fig. 5.10 Sequence node outputs when an object of w pixels moves to the right at speeds of $0.6w$, w, and $1.4w$ pixels per unit time.

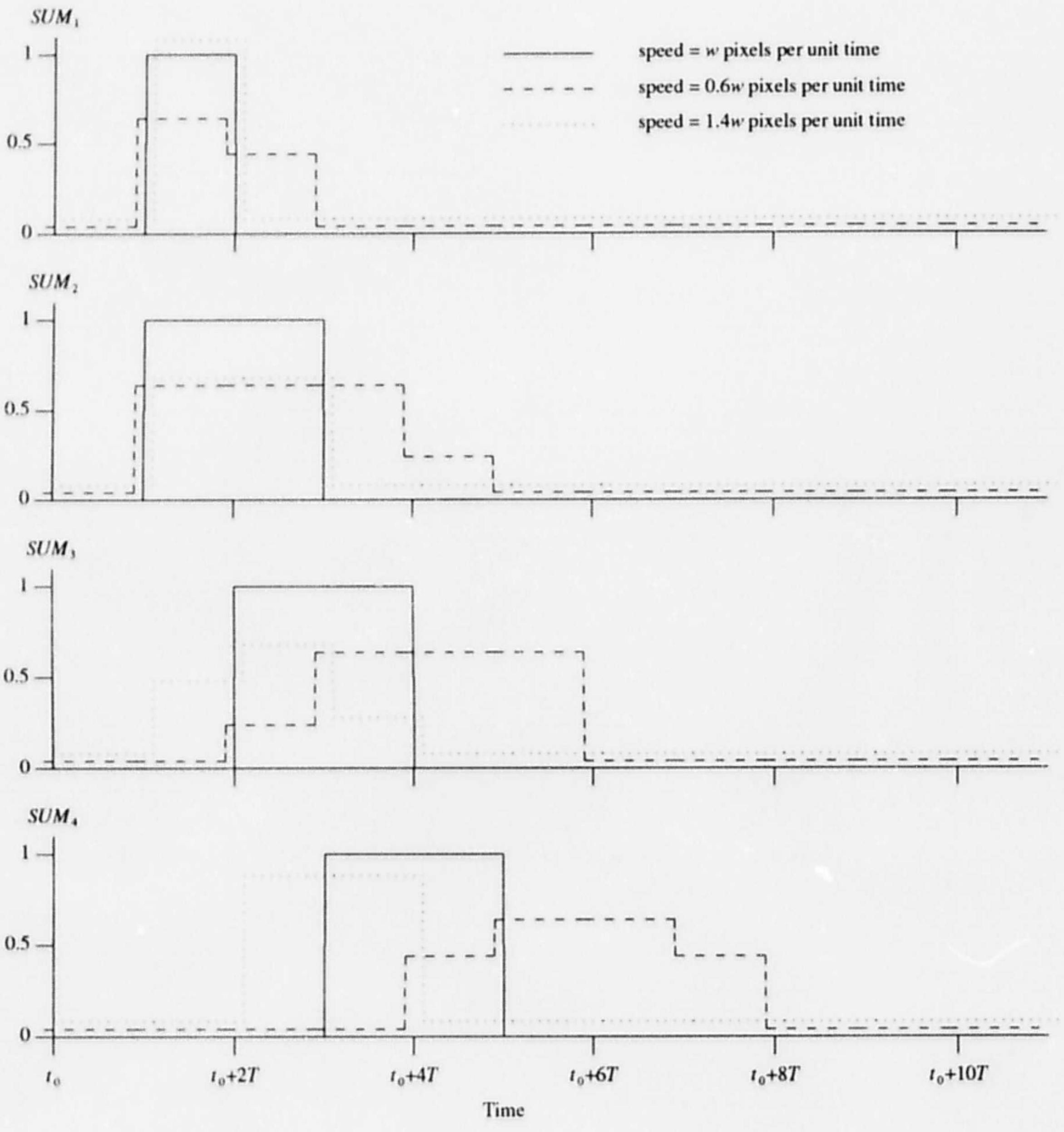


Fig. 5.11 Summer outputs when an object of *w* pixels moves to the right at speeds of 0.6*w*, *w*, and 1.4*w* pixels per unit time.

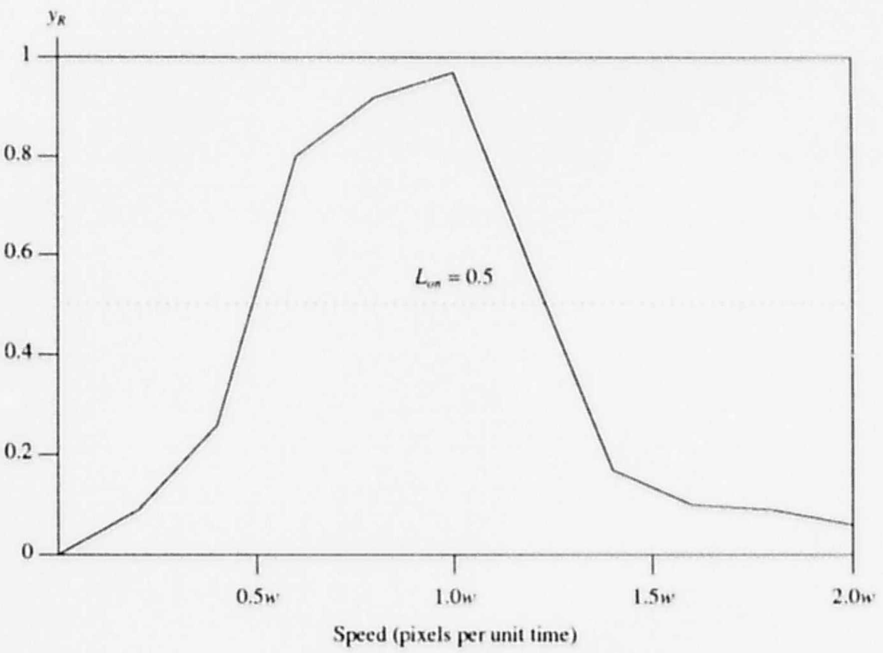


Fig. 5.12 Output of the motion detector at time $(t_0 + 11T)$ versus the motion speed.

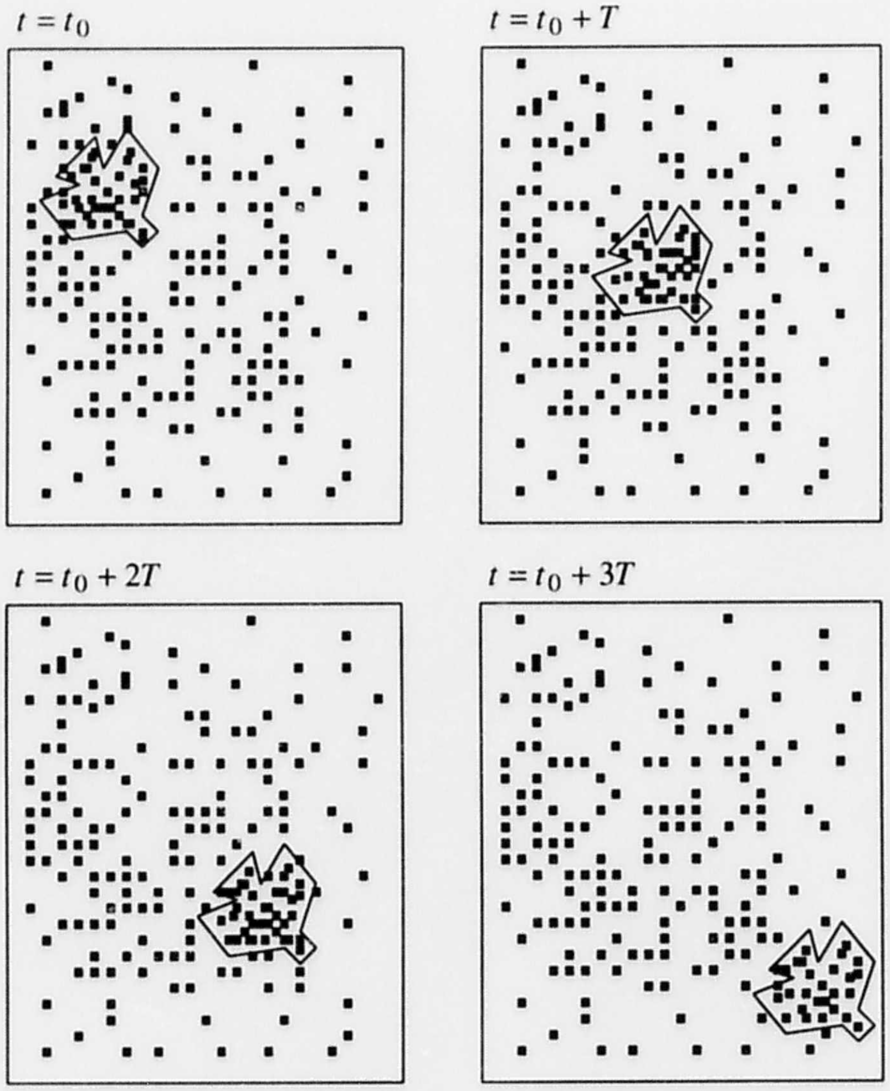


Fig. 5.13 A sequence of random-dot patterns.

difference between the two cases can be estimated by comparing the outputs of the summers during each period. Consider first the case of an object moving to the right at the training speed. Summer outputs for the movement of random-dot objects with the size of $0.4w$, w, and $1.8w$ pixels is presented in Fig. 5.14.

By comparing Fig. 5.8 with Fig. 5.14, it can be seen that, for an object size less than or equal to w pixels, the summers have similar responses in both cases, with a slightly lower amplitude for the case of random-dots. However, when the pattern size is greater than w pixels, the summers have similar responses most of the time, except for a few occasions. For example, the output of summer SUM_4 is less than 0.5 between time t_0+3T and t_0+4T when a solid object is presented to the detector, and greater than 0.5 when a random-dot object is presented. The reason is that in the case of a solid object, the movement of a large object may result in the current sample received by a summer being the same as the previous sample. The output of the summer during such a period is zero. In the case of an oversized random-dot object, the previous and the current samples received by a summer are different, because the arrangement of random dots in one part of the object is different from another part of the same object. Thus, a large number of the differentiators produce a nonzero output and the summer generates an output that is close to 1.

Consider now the case of a random-dot object moving at different speeds, where the size of the object is the same as the window associated with a summer. Fig. 5.15 presents the summer outputs when the motion speeds are $0.6w$, w, and $1.4w$ pixels per unit time. These curves can be compared with the summer outputs given in Fig. 5.11 for a w-pixel solid object moving at different speeds. Comparison shows that the summers have similar output behavior, except that the outputs in Fig. 5.15 are lower in magnitude than those in Fig. 5.11.

The movement of a solid object against a white background has a similar effect on the motion detector as the movement of a random-dot object against a random-dot background in most cases. Therefore, the results obtained from the performance study can be used to interpret the simulation results for larger networks, which will be presented in the next section.

5.7. Larger Networks

This section simulates and tests the operation of the motion detector in more detail using larger networks. Several motion detectors with two, four and six temporal modules have been implemented for single speed and multi-speed linear motion.

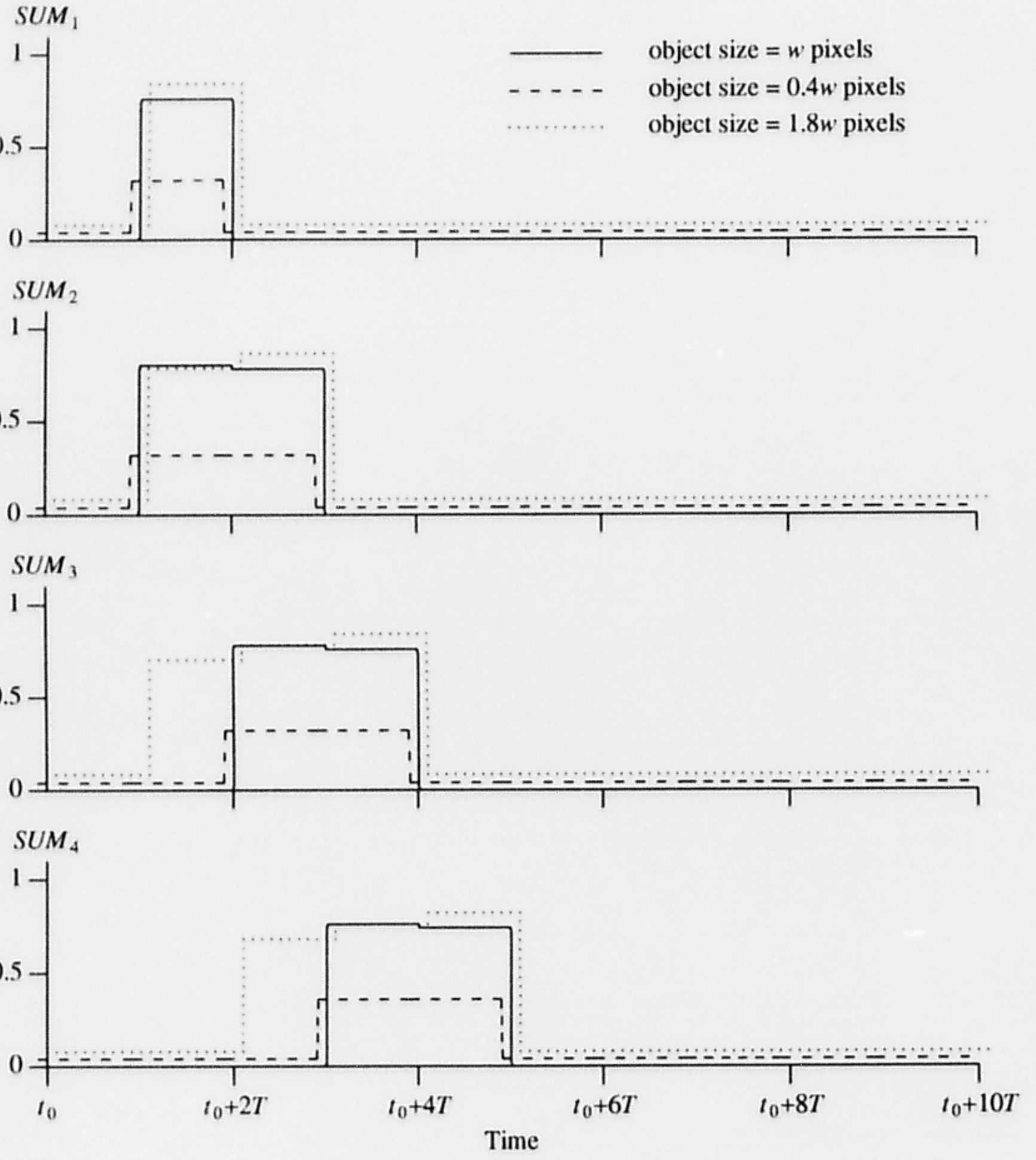


Fig. 5.14 Outputs of summers when a random-dot pattern with the sizes of $0.4w$, w, and $1.8w$ move to the right at a speed of w pixels per unit time over a random-dot background.

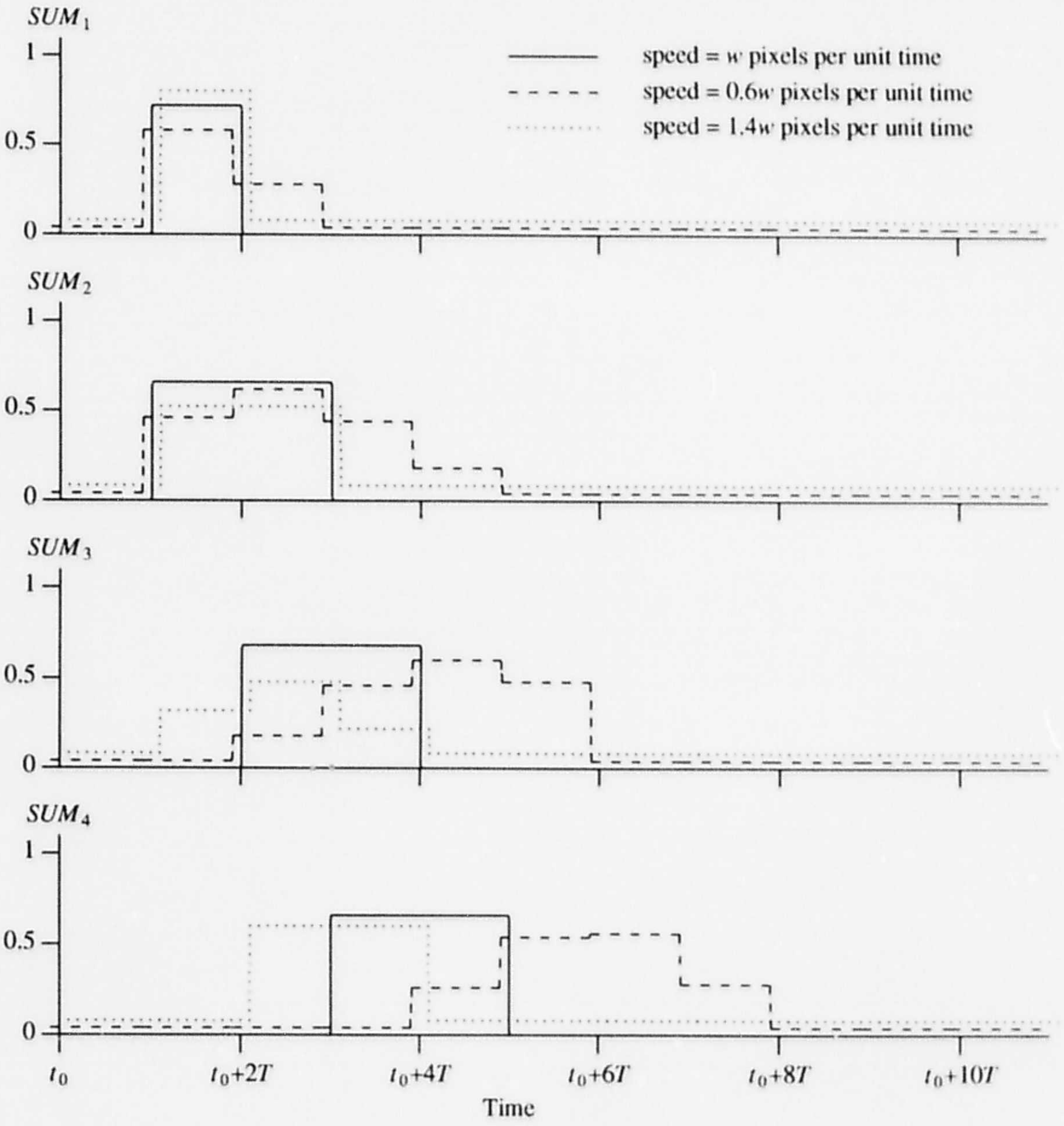


Fig. 5.15 Summer outputs when a random-dot pattern of w pixels moves to the right at speeds of $0.6w$, w, and $1.4w$ pixels per unit time over a random-dot background.

The image input used consists of an array of 160×32 binary pixels, and the window associated with each summer has the size of 8×32 pixels. The motion of an object is generated by moving a 8×32 pixels random-dot pattern against a 160×32 pixels background containing a different random-dot pattern. The sampling rate of the image input is $T = 10^{-4}$ second. A temporal module has 20 sequence nodes having a time constant of 10^{-4} second, and a result node with a time constant of 2×10^{-3} second. The initial weights of all the interconnections are equal to zero. A module is trained with a solid object and tested with a random-dot object.

Consider two interconnected modules, each trained to recognize the motion of an object at the same speed, but in different directions. Module 1 was trained to recognize an object moving left from one window to the next at a speed of 8 pixels per unit time, and module 2 was trained to recognize motion at the same speed in the opposite direction. Fig. 5.16 shows the performance of the motion detector for different speeds. For a training speed of 8 pixels per unit time, each module recognizes objects moving at speeds in the range of 2–10 pixels per unit time in the direction the module is trained. Note that these two curves have a similar shape as the one in Fig. 5.12. This confirms that the motion detector can detect the movement of an object at different speeds which are close to the training speed.

Another motion detector with two separate temporal modules was simulated. This time, each module was trained to recognize the movement of an object moving right, but at different speeds. Module 1 was trained to recognize motion at a speed of 8 pixels per unit time, and module 2 at 16 pixels per unit time. The outputs of modules 1 and 2 are shown in Figs. 5.17a and 5.17b, respectively, as a function of the speed of the test object. Module 1 can recognize objects moving at speeds in the range of 2–11 pixels per unit time, and module 2 recognizes in the range of 10–22 pixels per unit time. Note that the two modules are not interconnected.

Fig. 5.17c gives the performance of modules 1 and 2 when they are interconnected by inhibitory connections to operate in a winner-take-all fashion. Instead of being able to recognize objects moving at speeds in the range of 2–11 pixels, module 1 is now only able to recognize motion at speeds in the range of 2–10 pixels. Module 2 can recognize motion at speeds in the range of 11–20 pixels. The effect of the inhibitory interconnections between modules can be clearly seen in the sharp changes in response at the intermediate speeds of 10–11 pixels per unit time.

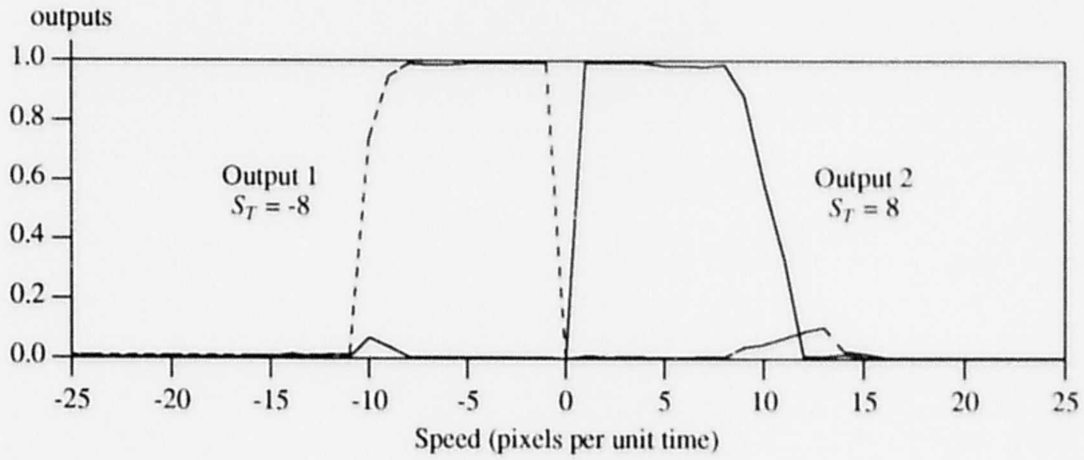


Fig. 5.16 Outputs of a motion detector trained at two speeds moving from left to right, or vice versa.

The case of a motion detector containing 6 interconnected modules each trained at a different speed is shown in Figure 5.18. Modules 1, 3 and 5 were trained to recognize an object that is moving left at a speed of 8, 16 and 24 pixels per unit time, respectively. Modules 2, 4, and 6 were trained to recognize motion of an object moving at a speed of 8, 16, and 24 pixels per unit time, respectively, but in the opposite direction. Modules 1 and 2 are able to recognize objects moving at speeds in the range of 2–10 pixels per unit time, modules 3 and 4 in the range of 10–20 pixels per unit time, and modules 5 and 6 in the range of 20–30 pixels per unit time. Note that for the speed range of 30–40 pixels per unit time, outputs of modules 5 and 6 do not fall sharply to zero because these two modules do not receive any inhibition from modules trained to recognize motion at a higher speed than their own.

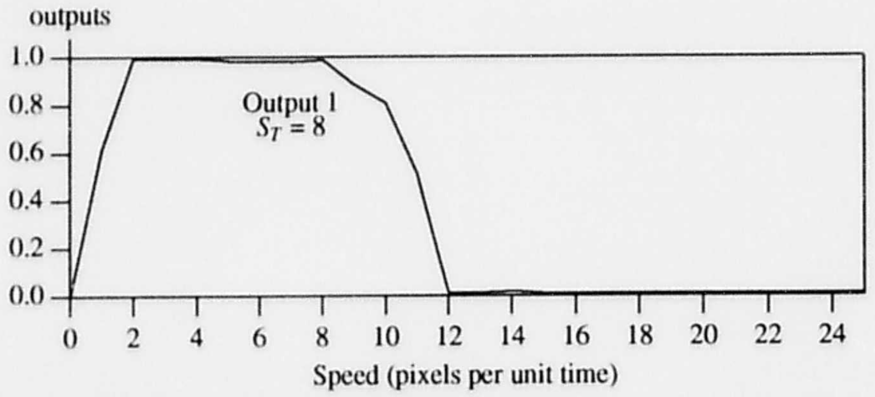


Fig. 5.17a Output of a module trained to recognize motion at 8 pixels per unit time.

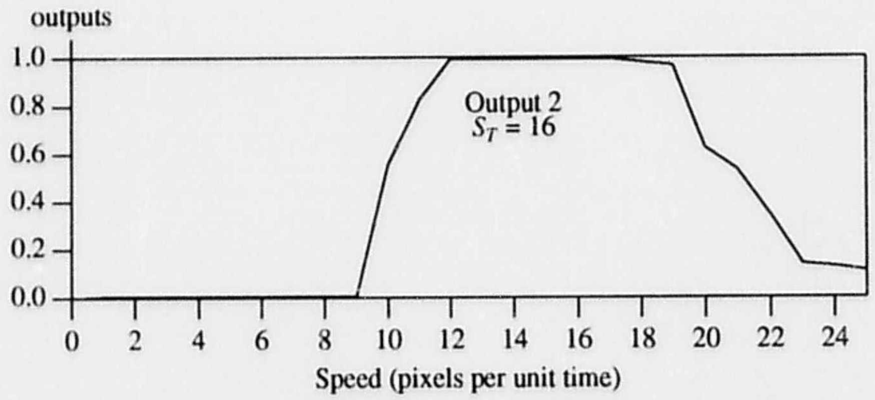


Fig. 5.17b Output of a module trained to recognize motion at 16 pixels per unit time.

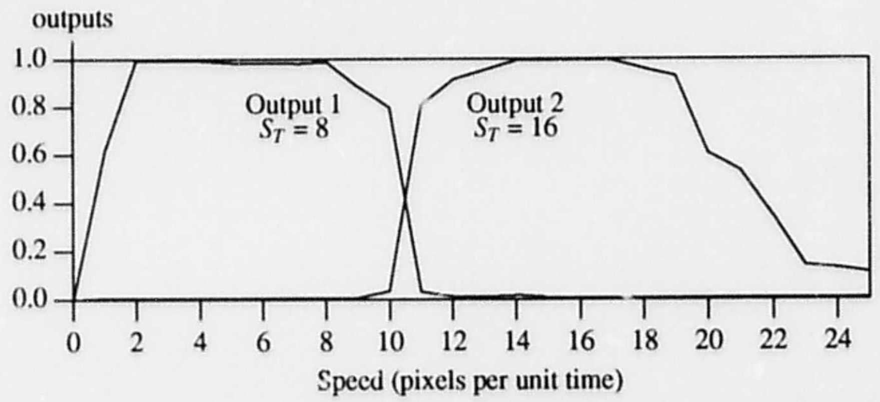


Fig. 5.17c Outputs of modules from above when they are interconnected by inhibitory connections.

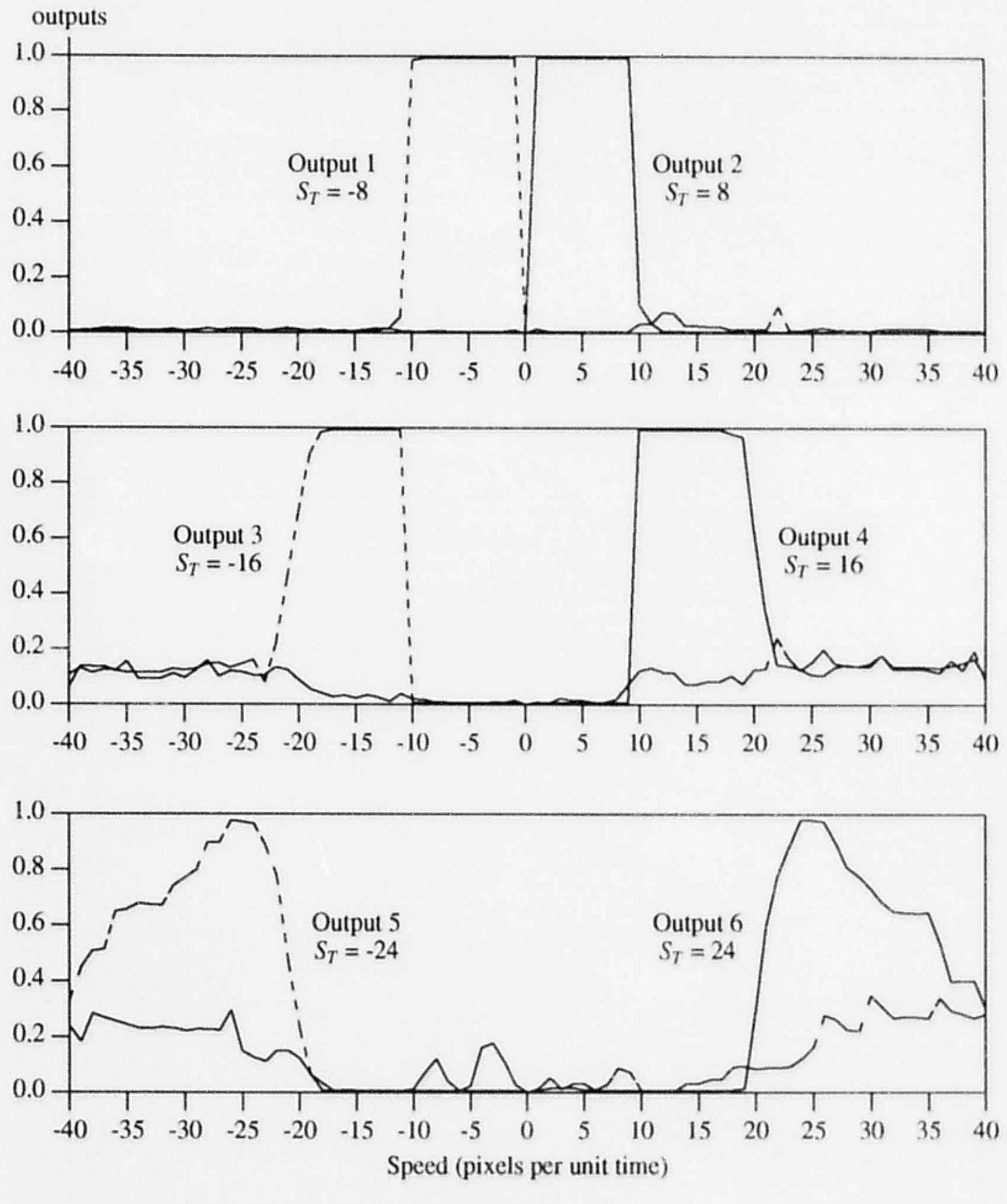


Fig. 5.18 Outputs of a motion detector trained at six speeds moving from left to right, or vice versa.

5.8. A Two-Dimensional Example

This section presents the simulation results of a 4-module motion detector trained to recognize multi-trajectory two-dimensional motion. It examines the effect on the performance of the detector of the object size, trajectory error, partial sequences, and input noise.

The image input used consists of an array of 80×80 binary pixels, and the input sampling period is $T = 10^{-4}$ second. The size of each window is 8×8 pixels. A module has 20 sequence nodes, each of which has a time constant of 10^{-4} second. The result node has a time constant of 2×10^{-3} second, and all interconnections have zero initial weights.

5.8.1. Trajectory

The trajectory of a moving object can be approximated by the edges of a polygon, an example of which is shown in Fig. 5.19. The solid line is the trajectory of a moving object and the dashed line is the training trajectory generated to approximate the actual trajectory, where a_0, a_1 and etc. are the sampling points. The approximation can be improved by increasing the number of sampling points along the trajectory. A training sequence of images depicting the movement of an object is generated by moving a square black box of the same physical dimensions as the object against a white back plane. The black box is placed at successive sampling points at successive time steps.

The generated sequence of images is then presented to the motion detector, one image at a time. The interconnection weights are adjusted in small increments using the LMS algorithm described in chapter 3.

An object moving at a certain speed will cause the generation of a sequence of images that is different from the sequence of images caused by the movement of the same object at another speed, even though the object moves along the same trajectory in both cases. An illustration is given in Fig. 5.20, in which T is the sampling period. The object on the left of the figure generates a sequence of images with a_0, a_1, a_2, ... etc. as the sampling points. The same object moving