It can be noticed that all the learning architectures discussed in this chapter, except the dynamic backpropagation algorithm[51], were originally formulated to address temporal sequence recognition tasks rather than learning control problems. In this chapter, the important link between the sensitivity model analysis, the recurrent backpropagation algorithms (by Almeida and Pineda), and the dynamic backpropagation algorithm, was discussed in detail.

It was pointed out earlier that the computational complexities involved in totally recurrent neural network architectures grow enormously as the number of network parameters increases. Also, some of the approaches require non–local information for the gradient estimation. The computing expenses associated with the dynamic backpropagation algorithm was discussed in the last section. In most of the cases, one model of the system is needed for estimating each of the partial derivatives $\frac{\partial e}{\partial \alpha_j}$. Such a scheme may not be suitable for real–time control applications. Narendra *et al.*[51] point out that static backpropagation networks may be sufficient in a number of adaptive control problems, provided the learning rate for the parameters are kept sufficiently small. Note that smaller learning rate results in poor dynamic characteristics of the system and obviously slower convergence. These aspects were discussed earlier in Chapter 4 and 5 and analyzed how the control architectures improve the dynamic response of the system.

All the recurrent network architectures discussed in this chapter, except the time–delay networks proposed in [72], use backpropagation as the learning algorithm. Even though some of the architectures ( for example, the schemes by Elman, Jordan and the memory neuron networks ) could be trained using any feedforward learning algorithm, in their original formulation backpropagation was used as the learning algorithm. Obviously, the RTRL algorithm[49], recurrent backpropagation algorithm[104][106] and the dynamic backpropagation scheme, are modifications of the static backpropagation learning to deal with dynamic signals and systems.

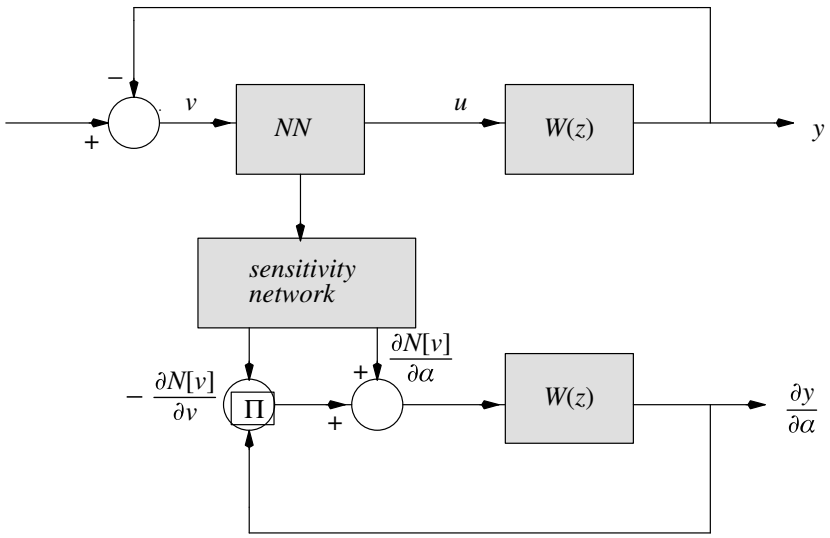



**FIgure 4.17 The Gradient Generation Scheme**

It can be noticed that all the desired partial derivatives used in the sensitivity model are generated using a single neural network (the sensitivity network). But each of this derivatives are to be processed through the dynamic system *W(z)*. Thus as many models of *W(z)* are necessary as the number of parameters of the network (weights). The computational complexity involved in such as scheme is enormous when the number of weights in the neural network is large. Narendra *et al.*[51] points out that the extent by which the output of *W(z)* differs from its input is the extent to which the dynamic gradient differs from that obtained using static backpropagation. Hence for systems of large time–delay or oscillatory pulse response, dynamic backpropagation is more valid than its static counterpart.

## 2.8. SUMMARY

In this chapter some of the important temporal neural network architectures and learning algorithms were reviewed in a unified fashion. Many variations to these approaches are possible, each having its own advantages ( and disadvantages ) in terms of computation, stability and convergence.

standard feedback control scheme, used in the studies discussed in this thesis (figure 5.16). In this case output $y$ is a function of $u$ while $u$ itself is a function of $y$. This is similar to a situation where a function $f$ is of the form $f[x(k,\theta_j),\theta_j]$. The function $f$ depends on the parameter $\theta_j$, both explicitly and implicitly. The total derivative of $f$ with respect to $\theta_j$ is given by,

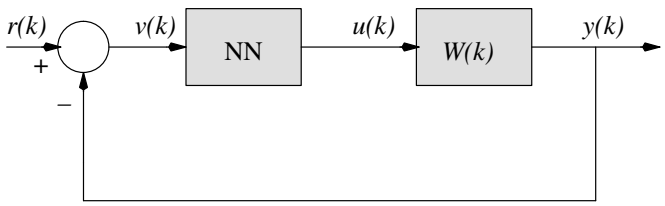



**Figure 4.16 Feedback Control Scheme**

$$\frac{\bar{\partial} f}{\bar{\partial}\theta_j} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial \theta_j} + \frac{\partial f}{\partial \theta_j} \tag{4.50}$$

In the scheme under consideration $y = N[v].W[z]$, and hence,

$$\frac{\bar{\partial} y}{\bar{\partial}\alpha_j} = \frac{\partial N[v]}{\partial v}\frac{\partial v}{\partial \alpha_j}W(z) + \frac{\partial N[v]}{\partial \alpha_j}W(z) \tag{4.51}$$

$v = (r–y)$ and hence

$$\frac{\partial v}{\partial \alpha_j} = -\frac{\bar{\partial} y}{\bar{\partial}\alpha_j} \tag{4.52}$$

Therefore the equation (6.51) reduces to

$$\frac{\bar{\partial} y}{\bar{\partial}\alpha_j} = -\frac{\partial N[v]}{\partial v}\frac{\partial y}{\partial \alpha_j}W(z) + \frac{\partial N[v]}{\partial \alpha_j}W(z) \tag{4.53}$$

The gradient generation scheme for scheme represented by equation (6.43) is shown in figure 6.17.

The objective is to generate the derivative of the error *e(k)* with respect to the network parameters (or weights) so that the parameters could be adapted to generate an optimum control signal *u(k)*. Since $y(k) = W(z)N(i)$ (where *N(.)* denotes the network i/o relationship), and $e(k) = y(k) - y_d(k)$, derivative of the error *e(k)*,

$$\frac{\partial e(k)}{\partial \alpha_j} = -\frac{dy(k)}{d\alpha_j} = -W(z)\frac{\partial u}{\partial \alpha_j} \tag{4.49}$$

The derivative $\frac{\partial u}{\partial \alpha_j}$ could be estimated from the static backpropagation network, considering the error propagating part of the network as a sensitivity network[51]. The overall gradient generation scheme for the representation shown in figure 6.14 is shown figure 5.15. It could be seen that the sensitivity model is structurally similar to the original control scheme.

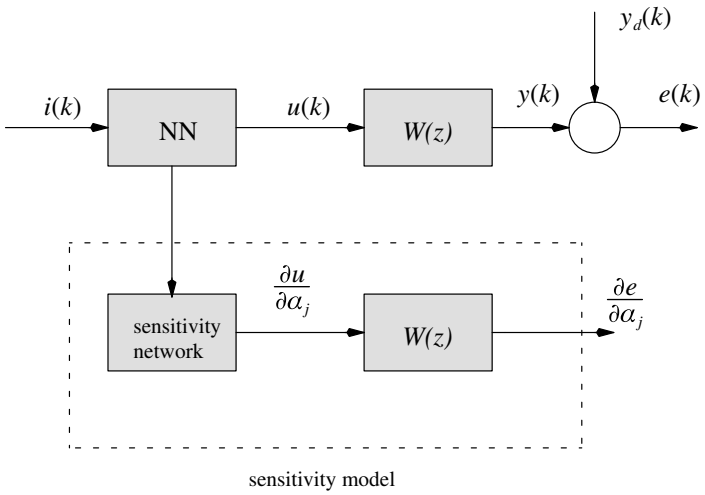



**Figure 4.15 Gradient generation scheme for the control scheme shown in figure 6.14.**

The above idea used for the system represented by figure 6.14 could be extended to any topology. Following are the relevant equations and the gradient generation scheme for the

is easy as the structure of the backpropagation network itself allows to generate derivatives of the of the output with respect to the parameters as well as the input to the network[51].

### 2.7.3. Gradient Methods using Backpropagation Networks

The previous two subsections dealt with the estimation of sensitivity models for linear as well as nonlinear systems. It should be noted that the sensitivity models remain architecturally the same as the original system. Only the inputs to the system are changed. We saw similar concepts in the studies by Almeida and Pineda that essential derivatives with respect to the neural network parameters could be generated by the network itself, irrespective of its architecture. These ideas are further extended by Narendra *et al.*[51][77] for backpropagation neural networks applied for system identification and control of dynamical systems.

Consider the following open loop control system (figure 5.14) in which a multilayer backpropagation neural network is used as the controller for the process $W(z)$. The output of the dynamics and the desired response, at each instant $k$, is represented by $y(k)$ and $y_d(k)$. Then the error at the instant k is $e(k) = y_d(k) - y(k)$. The performance criterion to be optimized, $J$, is a function of $e$ and is given by

$$J(\alpha) = \frac{1}{N}\sum_{0}^{N} e(k)^2 dt \qquad (4.48)$$

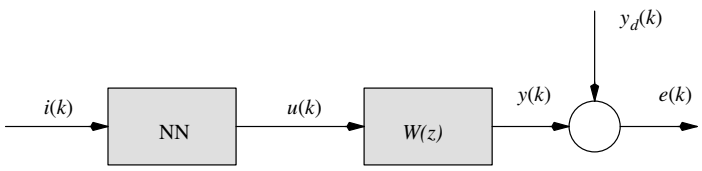



**Figure 4.14 Feedforward Control Scheme using Neural Network**

The sensitivity model is then described by rewriting the above equations as,

$$\ddot{z} + \frac{\partial F}{\partial \dot{y}}\dot{z} + z = -\frac{\partial F}{\partial \alpha}, \qquad where\ z = \frac{\partial y}{\partial \alpha} \qquad \text{and}$$

$$z(0) = \dot{z}(0) = 0 \tag{4.47}$$

If $\frac{\partial F}{\partial \dot{y}}$ and $\frac{\partial F}{\partial \alpha}$ are known, $\frac{\partial y(t)}{\partial \alpha}$ can be computed on–line using the system dynamics represented by equation (6.44). The sensitivity model will involve Jacobian matrices, if $y$ as well as $F$ are vector functions.

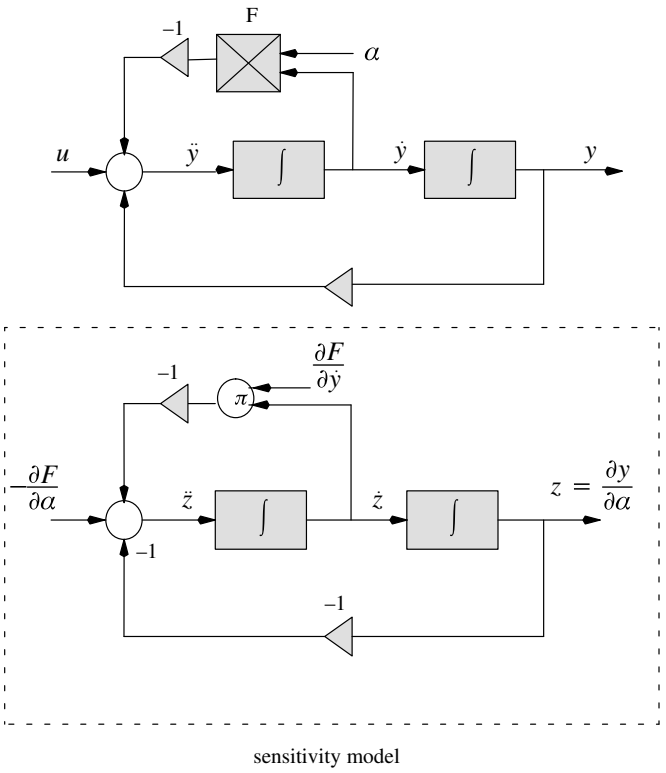



**Figure 4.13 Gradient generation scheme for the nonlinear system** $\ddot{y} + F(\alpha, \dot{y}) + y = u$

The gradient generation scheme for the above case is shown in figure 5.13. The nonlinear function $F$ should be known completely so that the required Jacobian matrix can be computed. When the nonlinearities are mapped into a neural network, computing the Jacobian

The generalized procedure described above can be applied to systems with more number of variables. In general, for each parameter in the system, the gradient is to be generated by a model described before and hence the procedure is more complex when the number of variables are more.

The generation of $\frac{\partial e(t)}{\partial \alpha}$ is based on the assumption that $\alpha$ is constant. Since $\alpha$ is adapted based on this very assumption, the stability of such a system cannot be ensured unless $\alpha$ is changed at a very slow rate. Or in other words, the learning rate $\eta$ should be sufficiently small, in such a scheme.

### 2.7.2. Gradient Methods for Nonlinear Systems:

In the case of nonlinear systems, the sensitivity models, like those derived in the case of linear systems, cannot be obtained directly from the equations describing the plant. The sensitivity models involve the estimation of the derivatives of the corresponding nonlinear functions. Consider the nonlinear function described by[77],

$$\ddot{y} + F(\alpha, \dot{y}) + y = u; \qquad y(0) = y_1 \;\&\; \dot{y}(0) = y_2 \tag{4.44}$$

As before, the objective is to optimize a squared error performance index given in equation (6.40.). The implementation of the gradient procedure once again involves the estimation of $\frac{\partial y}{\partial \alpha}$ over the interval *[0, T].*

Differentiating equation (6.44) with respect to $\alpha$

$$\frac{\partial \ddot{y}}{\partial \alpha} + \left[ \frac{\partial F}{\partial \dot{y}} \frac{\partial \dot{y}}{\partial \alpha} + \frac{\partial F}{\partial \alpha} \right] + \frac{\partial y}{\partial \alpha} = 0 \tag{4.45}$$

$$\frac{\partial y(0)}{\partial \alpha} = \frac{d}{dt}\left[ \frac{\partial y(0)}{\partial \alpha} \right] = 0; \tag{4.46}$$

$$\alpha(t) = \alpha(t - T) - \eta J_{\alpha}(\alpha) \tag{4.42}$$

Now our objective is to determine the parameter $\frac{\partial y}{\partial \alpha}$. Taking the partial derivative of equation (6.39) with respect to $\alpha$, we get;

$$\frac{\partial \ddot{y}}{\partial \alpha} + \alpha \frac{\partial \dot{y}}{\partial \alpha} + \frac{\partial y}{\partial \alpha} + \frac{\partial y}{\partial t} = 0 \tag{4.43}$$

Comparing equation (6.43) with (6.39), it can be seen that $\frac{\partial y}{\partial \alpha}$ is the output of the dynamical system described by $u = \ddot{y} + \alpha\dot{y} + y$ with an input $-\frac{\partial y}{\partial t}$. Hence if $\frac{\partial y}{\partial t}$ is available, $\alpha$ can be optimized with respect to a given performance index. A schematic representation of the gradient generation scheme for the above example is given in figure 5.12.

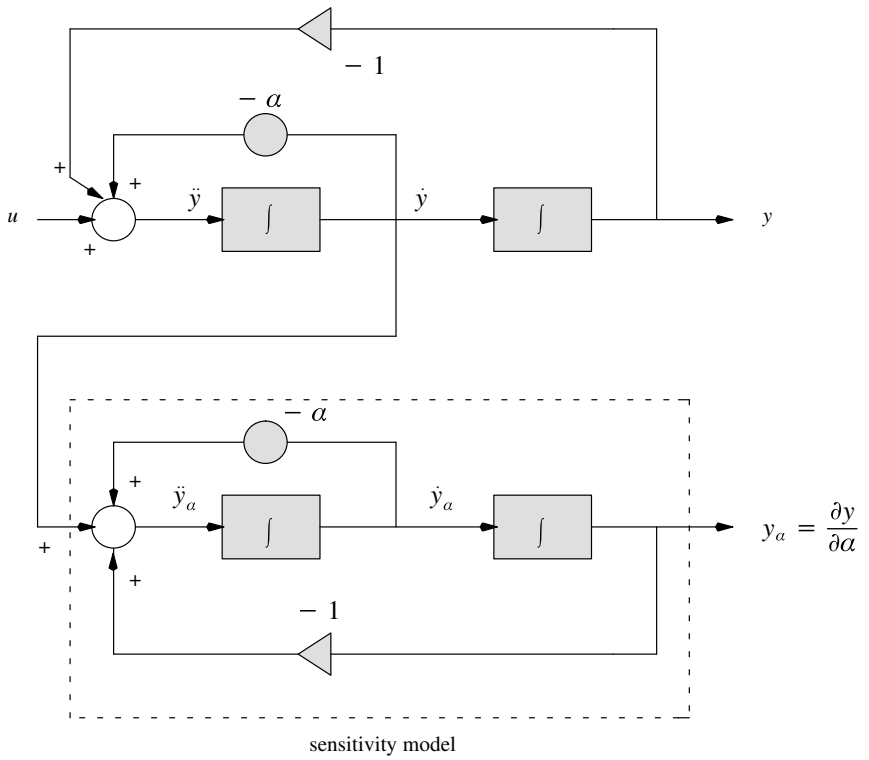



**Figure 4.12 Gradient Generation Scheme for the system** $\ddot{y} + \alpha\dot{y} + y = u$

ally the error propagation stage of the backpropagation neural networks is similar to the sensitivity models. Such an important link between the traditional adaptive control technique and the neural network learning, is very much useful in the application of neural networks on dynamic system identification and control. Some of the important results in this area are due to the studies of Narendra *et al.* in 1990[51] and 1991[77]. The dynamic backpropagation algorithm proposed in [51] and [77] treats the error propagation stage of the backpropagation networks as sensitivity networks producing the required derivatives needed for parameter adaptation. Detailed discussion on the parameter optimization techniques using sensitivity models, which are central to the above studies, could be found in [128–130].

### 2.7.1. Gradient Methods for Linear Systems

Consider the optimization of the parameter $\alpha$ in a linear dynamical system defined by[51],

$$\ddot{y} + \alpha\dot{y} + y = u \tag{4.39}$$

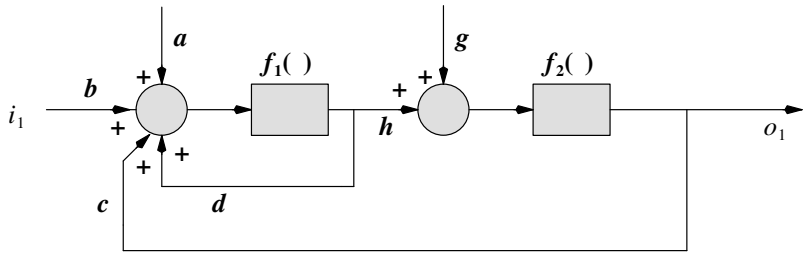
Let the performance function to be optimized is,

$$J(\alpha) = \frac{1}{T}\int_0^T [y(\tau) - y_d(\tau)]^2 d\tau \tag{4.40}$$

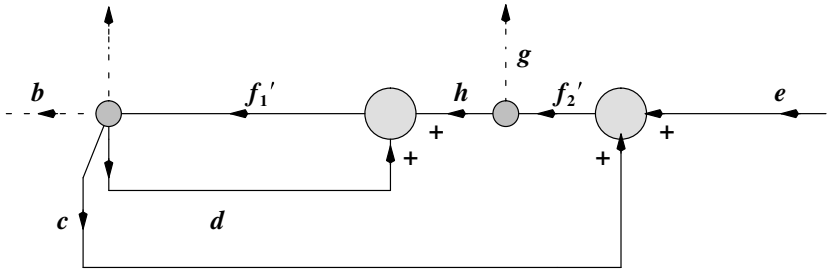
Then the derivative of the performance index with respect to the system parameter $\alpha$,

$$\frac{\partial J(\alpha)}{\partial \alpha} = J_\alpha(\alpha) = \frac{1}{T}\int_0^T 2[y(\tau) - y_d(\tau)]\frac{\partial y(\tau)}{\partial \alpha} d\tau \tag{4.41}$$

Hence the computation of the desired gradient involves the determination of $\frac{\partial y}{\partial \alpha}$. The optimization of the parameter $\alpha$ is performed by the following adaptation rule:




(a)




(b)

**Figure 4.11 (a)** **A simple recurrent network.**
**(b)** **The error propagation network of figure (a).**
**The dotted lines are not needed for the error propagation**

## 2.7. THE DYNAMIC BACKPROPAGATION ALGORITHM

It should be noted that one of the major contributions of the studies described in the previous sections (especially the works by Almeida, Pineda and Williams *et al*) is that they provided an important link between the optimization techniques used for dynamical systems in the context of adaptive control theory, and the neural network learning concepts. The sensitivity models, which generate the partial derivatives of the performance index with respect to the adjustable parameters of the system, were one of the widely used methods in the studies on the adaptive control system theory in the early 60's[128][129]. It can be seen that, conceptu-

and $$x_k'' = \sum_{j=i}^{N} t_{kj} net_j'' \qquad (4.38)$$

It could be seen easily that the transpose of the linearized perceptron network is a feedforward network and propagates in the reverse direction. This is essentially equivalent to the error propagating stage of the backpropagation algorithm[14]. Or in other words, it could be proved that the transpose of the linearized perceptron network is always the adequate network for the error propagation. Almeida[104] extends this idea to the feedback perceptrons in the following way: During the error propagation stage, consider the feedback as an additional error input to the neuron. Hence all the 'input' connections to a neuron becomes 'output' connections during the backpropagation of the error. For example, consider figure 5.11a; the neuron 1 has two feedback connections, one a self loop and the other one from neuron 2, and two other inputs *a* and *b*. The corresponding error propagation network is shown in figure 6.11.b. The input and output connections in figure 6.11.a. become, respectively, output and input connections in figure 6.11.b. The error propagation through the layers could be done as in the case of a normal backpropagation algorithm, taking the error contributions due to the feedback connections also.

Hence the whole procedure allows to extend the backpropagation algorithm to general architecture without the need for any non–local operations. The idea suggested by Pineda and Almeida allows the simultaneous operation of the original network and its transposed ( error propagation ) architecture. The feedforward network continuously supplies the error propagation network with the error signals and the error propagation network adjusts the weights in the feedforward network continuously. This is the basic idea behind the master–slave networks proposed by Lapedes and Farber[71].

$$net_i = \sum_j w_{ji} out_j + \sum_j t_{ki} x_k + \theta_i \tag{4.29}$$

where

$$out_i = f(net_i) \tag{4.30}$$

and $\theta_i$ denote the bias term. At the output layer, the output of neuron $p$,

$$o_p = out_p \tag{4.31}$$

The above equations are the equations of the equilibrium states of the network for a given input vector $\mathbf{x} = \{x_k\}$. If we linearize the network around the equilibrium state,

$$net_i' = \sum_j w_{ji} out_j' + \sum_k t_{ki} x_k' \tag{4.32}$$

$$out_i' = f'(net_i)\ net_i' \tag{4.33}$$

$$o_p' = out_p' \tag{4.34}$$

where the primes denote the variables of the linearized system. If we transpose the above and use double primes to denote the variables network is equivalent to the error propagation stage of the back propagation network. The transposed network is given by

$$out_i'' = \sum_j w_{ij} net_j'' + o_i'' \tag{4.35}$$

for the output neuron and

$$out_i'' = \sum_j w_{ij} net_j'' \tag{4.36}$$

for any other neuron; where

$$net_i'' = f'(net_i)\ out_i'' \tag{4.37}$$

to architectures having feedforward connections. Pineda[106] and Almeida[104] were the first to extend the backpropagation method to networks of any arbitrary architecture which converges to stable states. The power of their approach comes from treating the backpropagating stage as a modification of the feedforward network itself. They point out that the perceptrons with feedback are nothing but two feedforward networks unfolded in time. The error propagating part of the algorithm is treated as a separate feedforward operation unfolded in time. Such an approach could be applied to any feedback network as it could be unfolded in time into an equivalent feedforward one. This idea was further used in the studies on the application of neural networks for control of dynamical systems, by Narendra *et al.* [51][77]. The approach by Narendra *et al.* is discussed in the following section.

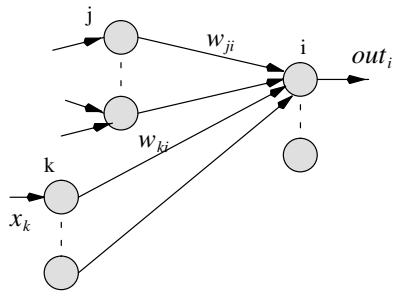



**Figure 4.10 A simple network architecture. The neuron *j* has input connections from the lower layer as well as from external inputs**

Consider a network as shown in figure 5.10. Each neuron has input connections from the previous layer and from the external input. Let $w_{ji}$ denote the weight matrix between the layers, $t_{ki}$ between the inputs and the layer neurons, $out_j$ the output of the neuron $j$ and $x_k$, the external input. Then the output of the neuron $i$,

In the last two equations, *f'* denotes the derivative of the nonlinear function *f*.

Similar network updating equations can be derived for the other weights in the network. Notice the computational complexity involved in training, compared to a feedforward network, even though we are dealing with only a simple two neuron network. It can be shown that for a general network, the change in weight $w_{pq}$, $\Delta w_{pq}$ could be written as

$$\frac{\partial y_i(t)}{\partial w_{pq}} = f'[\sum_j w_{ij} y_j(t-1) + I_i(t-1)][\delta_{ip} y_q(t-1) + \sum_j w_{ij} \frac{\partial y_j(t-1)}{\partial w_{pq}}] \tag{4.27}$$

where $\delta_{ip}$ denotes the Kronecker delta. An on–line updating of the weights are possible provided the learning rate $\eta$ is sufficiently small.

Equation (5.18) could be modified to generate networks of different dynamics and hence different learning rules. Pearlmutter[125] discusses a continuous–time recurrent network with a dynamics of the form

$$\tau_i \frac{dy_i}{dt} = -y_i + f(\sum_j w_{ij} y_j) + I_i(t) \tag{4.28}$$

and derives a set of updating equations in continuous time domain similar to the equations described earlier. This approach could be seen as an extension of the recurrent backpropagation (discussed in the following section) to inputs of sequential nature.

## 2.6. RECURRENT BACKPROPAGATION

It was pointed out earlier that backpropagation is one of the most popular learning algorithms for neural networks, but the original formulation of the algorithm was essentially restricted

$$e_k(t) = y_{d_k}(t) - y_k(t) \quad \text{if } y_{d_k}(t) \text{ is defined at time t} \tag{4.20}$$
$$= 0 \quad \text{otherwise}$$

The total cost function is then,

$$E(t) = \frac{1}{2}\sum_k e^2{}_k(t) \tag{4.21}$$

To derive an updating rule, we have to consider the the derivative of the output of the neurons with respect to all the weights in the network, as all the neurons are connected to all the other neurons in the network. Consider a fully recurrent network of two neurons as shown in figure 5.9. The output of the neurons are represented by the following equations:

$$y_1(t) = f[w_{11}y_1(t-1) + w_{12}y_2(t-1) + I_1(t-1)] \tag{4.22}$$

$$y_2(t) = f[w_{21}y_2(t-1) + w_{22}y_2(t-1) + I_2(t-1)] \tag{4.23}$$

If updating of one of the weights, say $w_{12}$ is considered, the change in the weight at each time instant $\Delta w_{12}$ is given by[14];

$$\Delta w_{12}(t) = \eta\ [e_1(t)\frac{\partial y_1}{\partial w_{12}} + e_2(t)\frac{\partial y_2}{\partial w_{i2}}] \tag{4.24}$$

where

$$\frac{\partial y_1}{\partial w_{12}} = f'[w_{11}y_1(t-1) + w_{12}y_2(t-1) + I_1(t-1)].[y_2(t-1) + w_{11}\frac{\partial y_1(t-1)}{\partial w12} + w_{12}\frac{\partial y_2(t-1)}{\partial w_{12}}] \tag{4.25}$$

$$\frac{\partial y_2}{\partial w_{12}} = f'[w_{21}y_1(t-1) + w_{22}y_2(t-1) + I_2(t-1)].[w_{21}\frac{\partial y_1(t-1)}{\partial w12} + w_{22}\frac{\partial y_2(t-1)}{\partial w_{12}}] \tag{4.26}$$

## 2.5. FULLY RECURRENT NEURAL NETWORKS: THE REAL−TIME RECURRENT LEARNING ALGORITHM (RTRL)

Williams and Zipser[49] introduced a generalization of the backpropagation algorithm, for recurrent networks with out the need of duplicating the units. This restricts the growing memory requirement associated with the backpropagation through time approach. In the RTRL approach, the equations governing the dynamics of the network could be written as:

$$y_i(t) = f(\sum_j w_{ij} y_j(t-1) + I_i(t-1)) \tag{4.19}$$

where $y_i(t)$ denotes the output of the neuron i $w_{ij}$, the interconnection strength between neuron $j$ and neuron $i$, $I_i(t)$ the external input to neuron $i$ and $f(.)$, a monotonically increasing nonlinear function. Note $w_{ij} \neq w_{ji}$ and the target output need not be defined at each instant in time $t$; that is the error measure on unit $k$ at time $t$ is defined by:

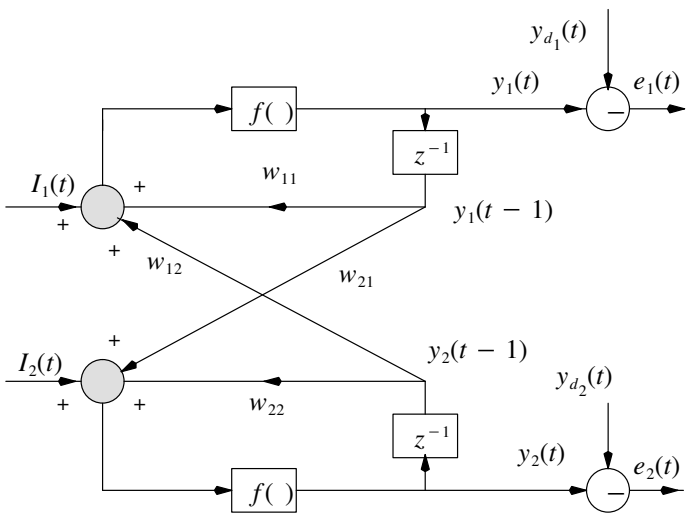



**Figure 4.9 A two neuron recurrent network, with the dynamics represented by equations (6.22) and (6.23)**

time steps is needed, it could be obtained by duplicating ( or unfold in time ) all the neurons $T$ times, so that separate neurons hold each of the states of the neurons over the period.

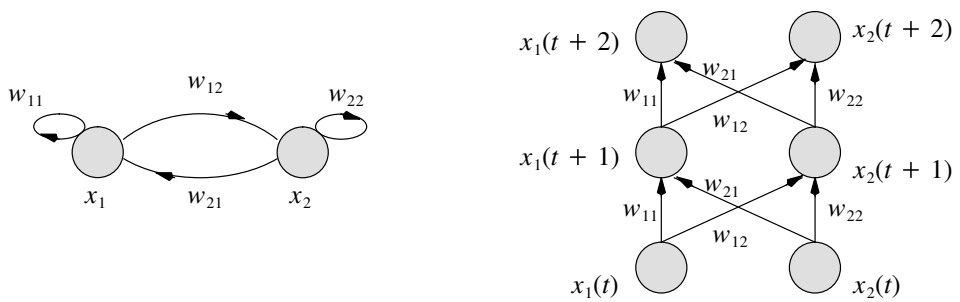



**Figure 4.8 Backpropagation through time network**

**(a) A two neuron fully recurrent network**
**(b) The equivalent feedforward network for three time steps**

It could be seen that any feedforward learning algorithm could be used for training such a network architecture. Note that when updating the weights layer by layer (as in the case of backpropagation algorithm) the change in 'copies' of the weights may not be equal. A proper averaging or normalization is needed in the change in weight, since for proper operation all the copies of the weights should be having equal values.

The drawbacks of the backpropagation through time approach are rather serious. Due to the duplication of the units and weights, the computational expenses and training time increases as the sequence lengths to deal with, is increased. For very long sequences and sequences of unknown lengths, such an approach would not be practically feasible. Because of the same reasons, this scheme is not a popular one and is not widely used.

for $k = 1,...,K$ where $K$ is the total number of memory neurons

$$y(t) = \sum_{k=0}^{K} W_k m_k(t) \tag{4.17}$$

The transfer function of the memory kernel, in discrete domain, could be expressed as

$$\frac{M_K(z)}{M_0(z)} = [\frac{\alpha z^{-1}}{1 - (1 - \alpha)z^{-1}}]^K \tag{4.18}$$

For stability $|1 - \mu| < 1$. Since the total memory structure is of the order K, the shape of the memory is not restricted to a recency gradient[110]. The parameters $K$ and $\alpha$ could be chosen appropriately to get an optimum performance.

## 2.4. BACKPROPAGATION THROUGH TIME (BTT) NETWORKS:

The network configurations discussed in the previous sections are partially recurrent, and the feedforward learning algorithms were used to train the networks. In the architectures proposed by Elman and Jordan, it was seen how time could be unfolded by adding extra neurons (context neurons). The same strategy could be applied to a fully recurrent neural network also[123]. This idea was originally proposed by Minsky *et al.*[6] was later extended using the backpropagation algorithm by Rumelhart *et al.*[14].

Consider a simple, two neuron fully recurrent network shown in figure 5.8a. Both the neurons have as self feedback as well as asymmetric connections between them. If we are interested in the behavior of the network over three discrete instants in time, it could be represented by an equivalent one as shown in figure 6.8b, which is feedforward in operation. Hence, if an equivalent feedforward network for a recurrent network operating through $T$

From equation (5.14) it could be seen that the memory convolution kernel is of first order. A better information storage capacity could be obtained with higher order memory kernel. The adaptive recursive memory networks ( or gamma networks ) proposed by de–Vries [110] uses such an approach. Figure 5.7 shows the information transfer between the regular neuron and the memory neuron in such a network. Even though the typical gamma networks are single layer feedforward networks, the theory could be easily extended to multilayer networks.

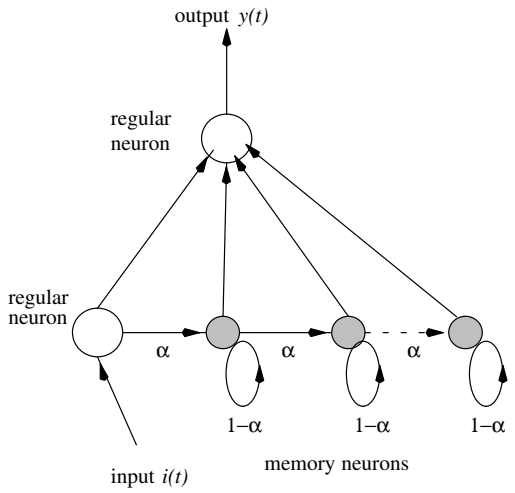



**Figure 4.7 Higher order memory kernel. The gamma network.**

Let us denote *o(t)* as the output of the regular neuron and $m_k(t)$ as the output of the $k^{\text{th}}$ memory neuron. Then, the network dynamics in the case of gamma networks (Figure 6.7) could be expressed as,

$$m_0(t) = i(t) \tag{4.15}$$

$$m_k(t) = (1 - \alpha)m_k(t - 1) + \alpha m_{k-1}(t - 1) \tag{4.16}$$

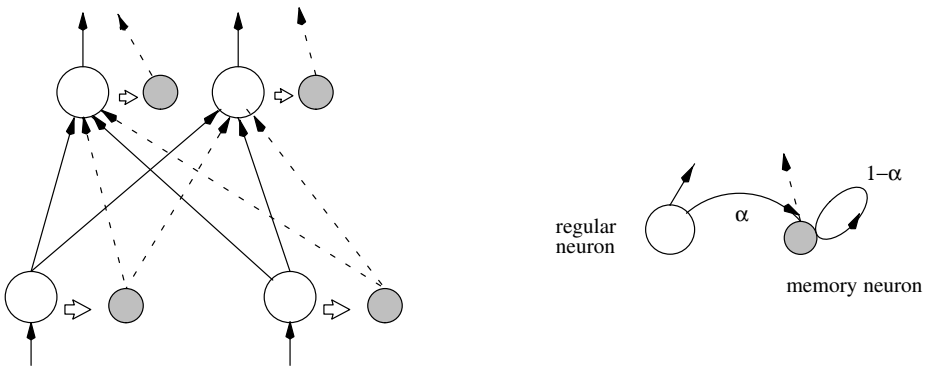



**Figure 4.5 Memory neuron network**

The backpropagation algorithm could be modified to incorporate the training for the interconnection strengths from a memory neuron also in a straight forward manner. In their implementation, Poddar *et al.* uses individual memorization coefficient which are adapted at each iteration. Also the learning coefficients for each weights are decoupled and adapted.

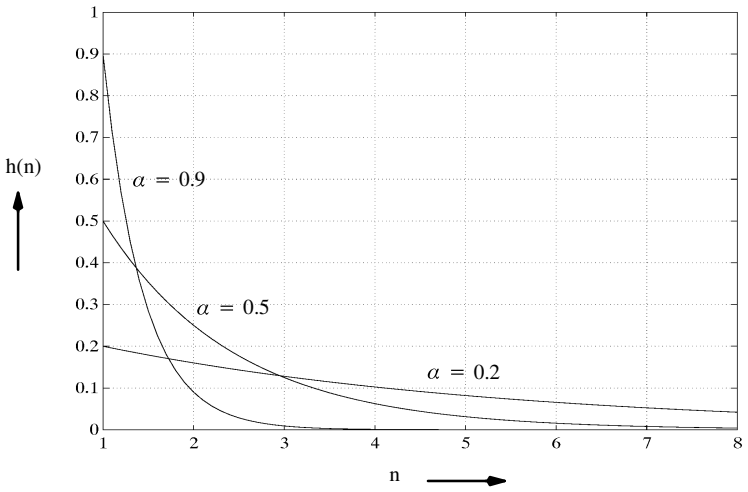



**Figure 4.6 Memory Kernel for different values of $\alpha$. The effective time window for the past neuron states becomes shorter as $\alpha$ approaches unity.**

connection strength from the memory neuron *j* in the lower layer to the network neuron *i* in the upper layer, and $out_j$ is the output of the neuron *j*. A nonlinear sigmoidal transformation of the form *f( )* to the neuron output results in,

$$out_i = f(net_i) = \frac{1}{1 + e^{-net_i}} \tag{4.11}$$

As in the case of a regular backpropagation neural network, the output of the neuron *'i'* is then given by a sigmoidal transformation.

The output of a memory neuron is a 'memorized' output of the network neuron according to the following equation,

$$m_j(t) = \alpha_j out(t - 1) + (1 - \alpha_j) m_j(t - 1) \tag{4.12}$$

where $\alpha$ is called the memorization factor or memory coefficient. If we represent the memory neuron dynamics in a discrete convolution form,

$$m_j(t) = h_j(t) * out_j(t) \tag{4.13}$$

where

$$h_j(t) = \alpha(1 - \alpha)^{t-1} \tag{4.14}$$

For stability $0 \leq \alpha \leq 1$. Figure 5.6 shows the convolution kernel for three different $\alpha$. For a higher $\alpha$, the network retains information for shorter duration, with higher strength. It could be seen that the contribution from the previous states of the neurons could be effectively controlled by the memory coefficient.

age of the its previous values and an $\alpha < 1$ ensures the stability of the system. One of the variations to this architecture is to have more than one context units, with different values of $\alpha$, for each output neuron.

A number of other architectures which are variations of the schemes by Elman and Jordan are proposed by other researchers. The schemes by Anderson *et al.*[107], Stornetta *et al.*[127] and Mozer[116] are some of them.

### 2.3.1. Memory Neuron Networks:

An alternate way to introduce dynamics into the network is to transfer the regular network neuron states to another set of *'duplication'* neurons called memory neurons, introduced by Poddar *et al.*[105][118] and de–Vries *et al.*[110][111]. Even though the architectures in these two cases differ from each other slightly, the underlying principle is the same. Figure 5.5 shows a multilayer perceptron architecture with memory neurons described by Poddar *et al.*[105]. The shaded circles represent the memory neurons and there is such a neuron associated with each of the '*network*' neurons (except in the output layer). The transfer of information to a memory neuron is only from its '*parent*' neuron but each memory neuron contributes information to all the *'network'* neurons in the higher layers. Each memory neuron has a self feedback loop, which stores the previous states. The dynamics of such a network could be expressed as,

$$net_i(t) = \sum_j w_{ji} out_j(t) + \sum_j x_{ij} m_j(t) \qquad (4.10)$$

where $net_i$ is the summed output of the neuron $i$, $w_{ji}$ is the interconnection strength from the network neuron $j$ in the lower layer to the network neuron $i$ in the upper layer, $x_{ij}$ is the inter-

between the input neurons to hidden layer and between the hidden layer and the output layer, respectively. Since the context units are treated as inputs, the network could be trained using the conventional backpropagation algorithm.

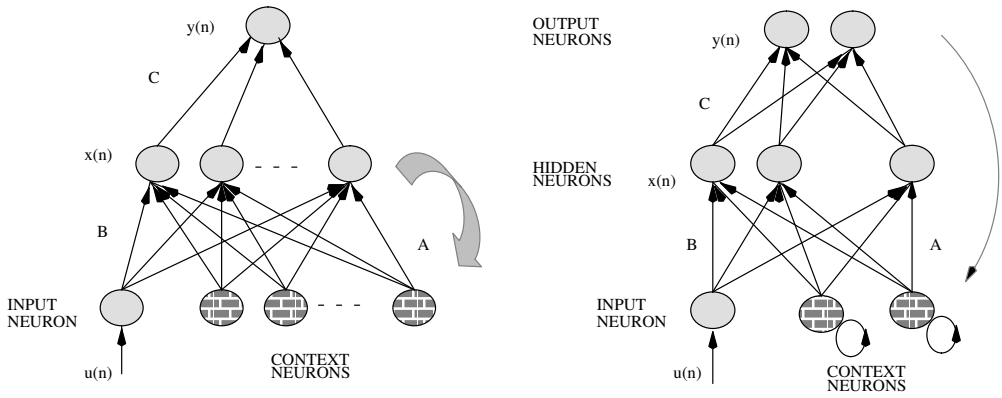



**Figure 4.4 Partially recurrent network architectures.**

**(a) Feedback from the hidden layer to the input layer[62]**
**(b) Feedback from the output layer to the input layer[113]**

Figure 6.4b shows the 'sequential architecture' proposed by Jordan[113]. The context units are copies of the output layer ( rather than that of the hidden layer as in Figure 6.4a) and has self feedback loops on themselves. The state equations for this architecture could be written as,

$$\begin{aligned} \bar{x}(n) &= A\bar{z}(n) + Bu(n) \\ \bar{y}(n) &= C\bar{x}(n) \end{aligned} \tag{4.8}$$

where,

$$\bar{z}(n) = \alpha\bar{z}(n-1) + \bar{y}(n-1) \tag{4.9}$$

represents the output of the context units. The parameter $\alpha$ is the connection strength of the self feedback loop. The context units could be shown to accumulate a weighted moving aver-

mum is a measure of the evidence of occurrence of the corresponding sequence as an input to the network. Because of the characteristics of the delays, even when the input sequence is time distorted, detection of the sequence is possible. Adding additional constraints to the energy function could improve the surface of the energy function so that a detection with better accuracy is possible when a sequence has occurred in the input stream.

## 2.3. NEURAL NETWORKS WITH MEMORY

A simple way to build a dynamical neural network architecture incorporating feedforward connections is to introduce the feedback information as the input. The feedback connections are fixed, with no adaptation involved, there by avoiding the stability issues associated with recurrent networks. Each neuron in the network need not have feedback information as inputs. The feedback connections could be from output layer to the input layer [113], from the hidden layer to the input layer[62] or from the neurons to themselves[114]. Two of such architectures are shown in figure 6.4. Note that a number of combinations are possible and all of them are subsets of the class of totally recurrent architectures.

Figure 5.4a shows the scheme suggested by Elman[62]. The input layer has two types of neurons, the context neurons and regular neurons. The context neurons are the copies of the hidden neurons in the previous time step. Hence there are as many number of context neurons as the hidden neurons. The state equations for such a network could be expressed in the form:

$$\left.\begin{aligned} \bar{x}(n) &= A\bar{x}(n-1) + Bu(n) \\ \bar{y}(n) &= Cx(n) \end{aligned}\right\} \tag{4.7}$$

where $A$, $B$ and $C$ denote the weight matrices between the context units and hidden layer,

### 2.2.1. The Time−Delay Network Dynamics

In the formulation used in static neural network dynamics, obviously the energy measure $E$ is not a function of time. However, in a time–dependent domain, $E$ is a function of the neuron states (output of the neurons) and time. That is,

$$E = E(\vec{o}, t) \tag{4.3}$$

Taking the derivative with respect to time;

$$\frac{dE}{dt} = \frac{\partial E}{dt} + \Delta_o E \frac{d\vec{o}}{dt} \tag{4.4}$$

Let us choose a nonlinear transfer function of the form $\vec{o} = f(\vec{i})$, where $i$ is the summed input to the neuron. Then if we let the circuit dynamics to be

$$\frac{d\vec{i}}{dt} \propto -\Delta_o E \tag{4.5}$$

equation (6.4) gets modified as,

$$\frac{dE}{dt} = \frac{\partial E}{\partial t} - (f^{-1})' \frac{d\vec{o}}{dt}^2 \tag{4.6}$$

Contribution produced by the dynamics of the network (the second term in equation (6.6)) is always positive as long as the transformation function used is monotonically increasing. At each instant, the network is forced to a valley in the time space.

An energy function $E$ can be created such that at the end of a sequence, the total state energy of the network reaches a minimum (a valley) on the energy surface. The depth of the mini-

n=5, plotted for different values of *k.* One of the advantages of this approach is that it works in continuous time domain, and there is no need for an external synchronization clock[72].

In the formulation first suggested by Tank and Hopfield, the connection matrix ( or the network weights ) are hard–wired. The connection strengths ( binary, 0–1) were hand picked according to the sequences to be detected. An extension of this scheme in which the connection strengths are trained using a training set, is described by Unnikrishnan *et al.*[72]. In this scheme, a feedforward network is used for the classification, rather than the Hopfield type of network used in [124]. Also, they found that Gaussian delay kernels show better results than the kernel represented by equation (6.2). Note that the purpose of neural network in the above cases is only for the detection of the sequence. The feature extraction is not a part of the neural network operation.

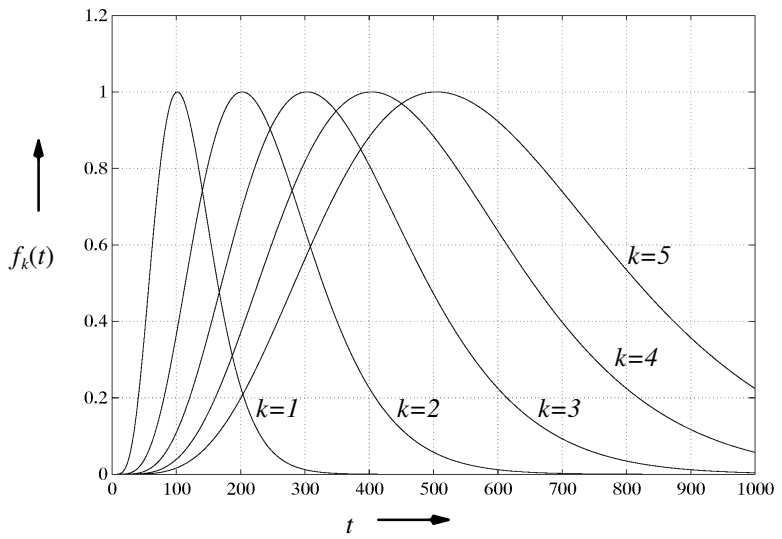



**Figure 4.3 The delay kernel for n=5**

of the feature detectors, form the input to the time delay connections. Input to the network can be expressed in the form:

$$I_i = \int_0^{\infty} f_k(\tau) D_X(t - \tau) d\tau \tag{4.1}$$

where $D_X(t)$ denotes the function representing the input events and $f_k(t)$ is the delay function. The function $D_X(t)$ is defined as;

$$\begin{aligned} D_X(t) &= 1 && \text{if input event } X \text{ is present at time } t \\ &= 0 && \text{otherwise.} \end{aligned}$$

The delay function is chosen as

$$f_k(t) = (\frac{t}{k})^n e^{n(1-\frac{t}{k})} \tag{4.2}$$

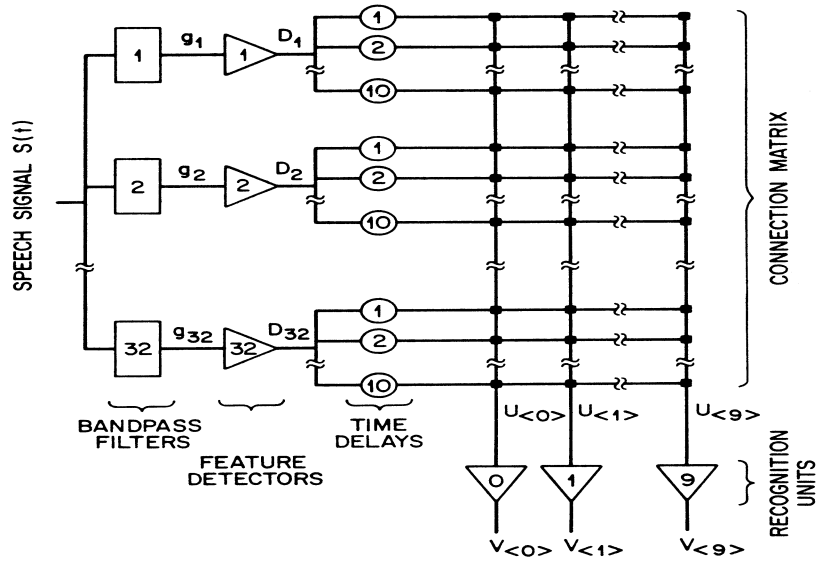



**Figure 4.2 Time delay network for speech recognition, proposed by Tank and Hopfield[124]. Adopted from [72].**

which is normalized to have a maximum value 1 when $t = k$. The parameter $n$ controls the width of the delay. For larger $n$, the function has a narrower peak. Figure 5.3 shows $f_k(t)$ for

used for the recognition of speech signals[62][63][124]. An extension to this scheme is to introduce time delays in the higher layer neuron inputs also[48]. The hidden neurons and output neurons get delayed inputs from the lower layer neurons. Such a scheme is shown in figure 6.1b Both schemes are generally termed as time delay networks.

There are several drawbacks to this general approach. A long sequence requires more delay lines and hence networks of larger size. Sequences of arbitrary length can not be accommodated. The input signals have to be properly registered in time and should arrive at exactly the correct rate. Also, this approach does not distinguish relative temporal position from the absolute temporal position[62].

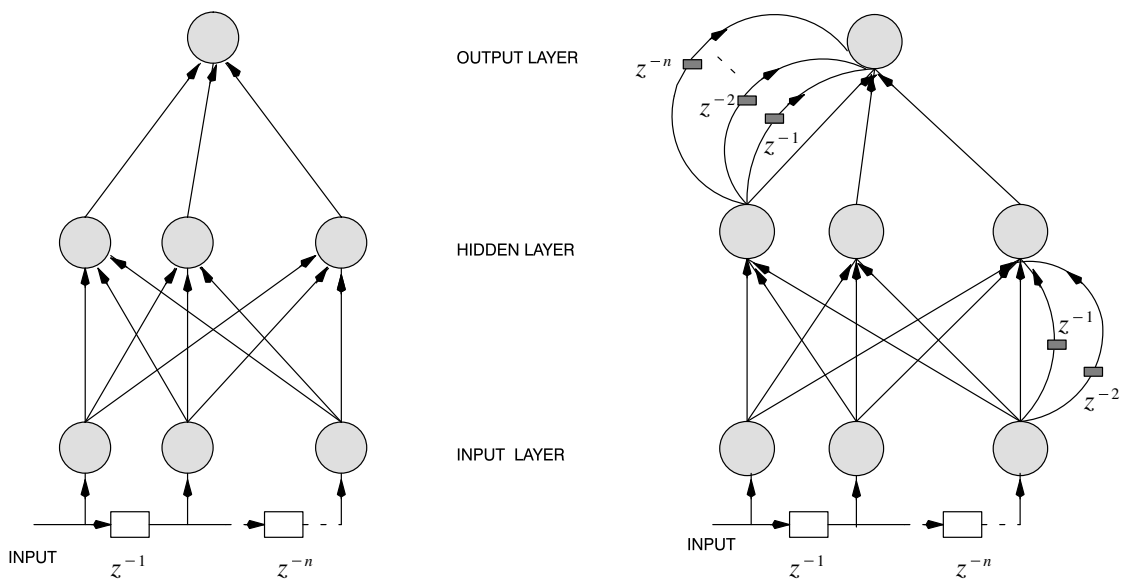



**Figure 4.1 Time−delay Neural Network**
**(a) Tapped delay lines for the input**
**(b) Time−delayed interconnections**

To accommodate input sequences of arbitrary length, Tank and Hopfield[124] suggest replacing the fixed delays by filters which broaden the signal in time as well as delaying it appropriately. Such a scheme is shown in figure 5.2. The signals *D1, D2,* .. which are the output

industry market predictions[123]. There are two distinct methods by which the above connectionist approach to the dynamical systems are addressed. The first approach is to design an appropriate dynamic architecture for the network. The studies by Elman[62], Jordan[113], Rumelhart *et al.*[14] and Werbos[123] belong to this category. Other examples are the memory neuron networks by Poddar *et al.*[105], the gamma networks by de–Vries *et al.*[110][112] and the time delay network architectures[48][72][124]. The problems associated with time delay networks are many, and will be addressed in one of the following sections. The memory neuron approach tends to be a better architecture because of the distributed representation of the memory and the elimination of the restriction on the input sequence length. The second method is to modify the learning algorithm itself to accommodate the dynamics into it. The real–time recurrent learning algorithm by Williams *et al.*[49], recurrent backpropagation algorithm by Almeida[104], Pineda[106], the modified backpropagation algorithm by Sudharsanan *et al.*[81] and the dynamic backpropagation by Narendra *et al.*[77] are examples for this approach. Even though almost all of the reported results are the extensions of the backpropagation algorithm, the idea could be extended to any learning algorithm.

## 2.2. PARALLELIZING THE TIME AND TIME DELAY NETWORKS

One of the straight forward approaches to deal with temporal information is to represent time as an explicit dimension of the input vector space. The first temporal event is represented by the first element, the second event by the second element and so on$_s$. Such a scheme is shown in figure 5.1a. Each neuron in the input layer gets information corresponding to a particular time instant. The entire input vector is then processed in parallel[62]. This scheme, generally called parallelizing the time approach or tapped delay line scheme, has been widely

ploited only if the networks themselves are dynamical in nature. Although, the number of learning algorithms and network architectures described in the literature for applications involving dynamical systems are many[14][26][47–51][62][72][75][77][81][92][104–126], they may be classified as belonging to one of the two distinct approaches. The approaches are (i) Building a recurrent network architecture and using a static learning algorithm for training; and, (ii) Implementing a dynamic learning algorithm. The first approach can be seen as a subset of the second one, but their capabilities and complexities differ significantly. This chapter gives an overview of both the approaches and attempts to view the recurrent neural network models and algorithms from a single global frame work.

## 2.1. IMPORTANCE OF TIME AND ITS REPRESENTATION

Time is an important parameter when dealing with dynamical systems. The temporal extension of behavior is present in all the living organisms[62]. The information flow in biological neurons always have a 'time tag' and significant time delays are present in neural system synapses and axons[124]. But representing time and its influence on the behavior of the system (whether living or artificial) has been one of the most difficult challenges in applications involving vision, modelling and control. Even within the traditional serial frame work of computation, representation of temporal information and its influence on system behavior is a challenging problem[62]. In the case of connectionist models, as the serial nature of the temporal events often contradicts with the parallel nature of computation, representation of time is more complex and challenging.

A large number of applications of neural network techniques to dynamical systems have been reported. They range from standard applications such as speech recognition [48][72][124], system identification[51][81[82][126], learning control [51][62][114] to gas

# CHAPTER 2.

# RECURRENT NEURAL NETWORK ARCHITECTURES AND ALGORITHMS FOR APPLICATIONS INVOLVING DYNAMICAL SYSTEMS

In the previous chapters, the discussions were confined to the feedforward neural networks and their applications involving dynamical systems. A vast majority of the existing neural network approaches for static pattern recognition, system identification and learning control tasks, use feedforward network architectures and static learning algorithms because of their simpler and well understood training schemes and the easiness in implementation. Also, they have the additional advantage of having better stability properties and better analyzable convergence characteristics.

Even though under restricted operating conditions, static networks perform well, in many of the dynamic applications, the real computing power of connectionist models could be ex-

The main reason for this reduction is the recurrent network architecture employed when the Alopex algorithm is used. As previously mentioned, the backpropagation algorithm cannot be used directly on a recurrent network architecture.

Another important advantage of the Alopex algorithm is its ability to extract information from noisy data. In Chapter 3, in the context of the MONKS problems, we demonstrated how Alopex can outperform backpropagation, when the input data is noisy. Further in figure 7.10, it is shown that the Alopex algorithm is effectively insensitive to measurement noise when used in learning control applications. The results of using the backpropagation algorithm for the same task is also depicted in figure 7.11. In the case of the backpropagation controller, the mean square error measure increases approximately linearly as the noise content in the signal is increased, where as in the case of the Alopex controller, the error remains as a smaller constant.

## 7.5. SUMMARY

In this chapter, the Alopex algorithm is projected as an effective learning algorithm for neural networks having asymmetric feedback connections and memory neurons. It is shown that the memory neuron networks trained using the Alopex algorithm can closely model nonlinear dynamical systems. Using the nonlinear AUV control problem as the example, it is found that recurrent neural networks trained using Alopex algorithm perform very well as learning controllers. Their tracking performance and stability properties are found to be equally as good as the backpropagation based controllers presented in Chapter 4. Further, it is observed that the Alopex controller has better measurement noise rejection capabilities than the backpropagation controller. This supports our earlier conclusions on the generalization capabilities of the Alopex (Chapter 3), based on its performance with static pattern recognition tasks.

disadvantages of the Alopex algorithm for on–line learning control applications is that the instantaneous updating of the network weights is undesirable. This necessitates the incorporation of a batch updating procedure which in turn increases the response delay of the controller. The batch size has to be decreased whenever there is a sudden change in the command signal or in other system parameters.

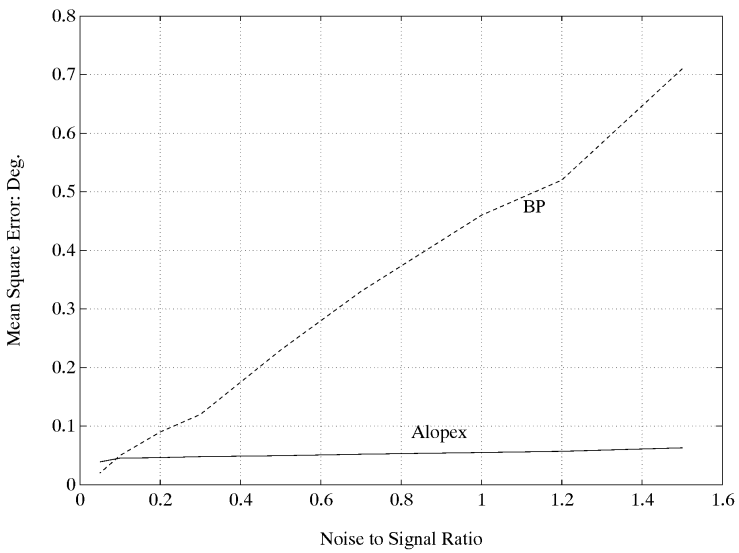
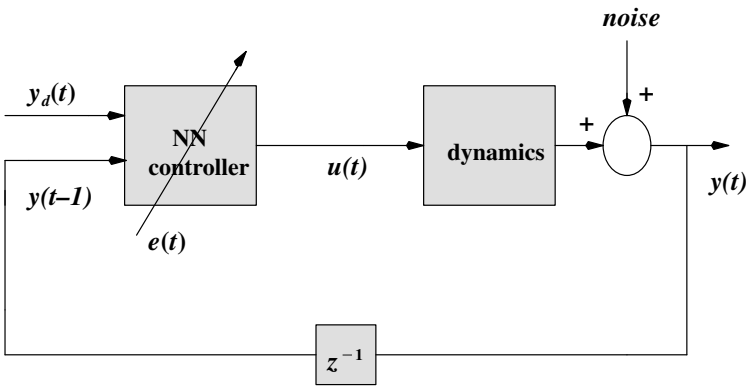
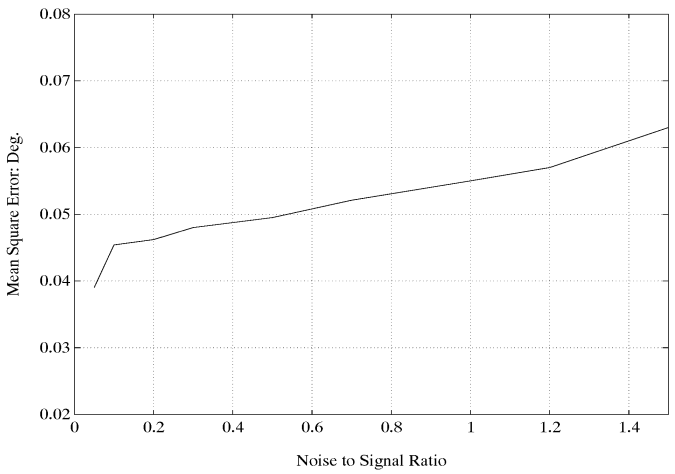



**Figure 5.11 Comparison of Backpropagation and Alopex. Mean Square Error plotted as a function of the Noise to SIgnal Ratio.**

An analysis of the results using backpropagation (Chapter 4) and the Alopex algorithm on the AUV control problem, provides the conclusion that tracking performance and stability characteristics of the Alopex controllers are equally as good as those of the backpropagation controllers. However for the same level of performance, more complex networks are required when backpropagation algorithm is used. For studies described in this chapter, Alopex networks of size 2x10x1 (141 interconnections) are used, while the network size used in the studies with backpropagation are 2x20x10x1 (281 interconnections). Thus the Alopex networks needed only 50% fewer interconnections than that for backpropagation networks.




(a)




(b)

**Figure 5.10 Controller performance under measurement noise**
**(a) Injection of white noise at the output**
**(b) Mean square error plotted as a function of noise to signal ratio**

## 7.4. COMPARISON OF BACKPROPAGATION AND ALOPEX ALGORITHMS

Along with the experimental studies on static pattern recognition tasks, some salient features of the Alopex algorithm were discussed in Chapter 2. Those properties hold good even when the algorithm is used for system identification and learning control tasks. One of the major

duced by 50% instantly at 2000 sec. It can be seen that network is able to adapt accordingly to drive the vehicle to the specified pitch. Another parameter having significant importance in the controller performance is the measurement noise. A low signal–to–noise ratio(SNR) may create poor tracking and even instability in the case of traditional controllers. One of the promising properties of the neural networks is their ability to generalize even when the input data is noisy. In order to study the ability of the neural network controller to adapt to the measurement noise, we introduced a white noise component into the measured response (pitch) of the vehicle (see figure 7.10a). The noise amplitude is varied from 5% to 150% of the output signal and the mean square error between the desired and actual responses is calculated. The mean square error plotted as a function of the noise to signal ratio is shown in figure 7.10b. It can be observed that the variation in the error is very small (only about 0.025 degrees) for a substantial change in the noise–to–signal ratio. This shows that the controller is able to reject the noise very effectively over a wide range of levels.

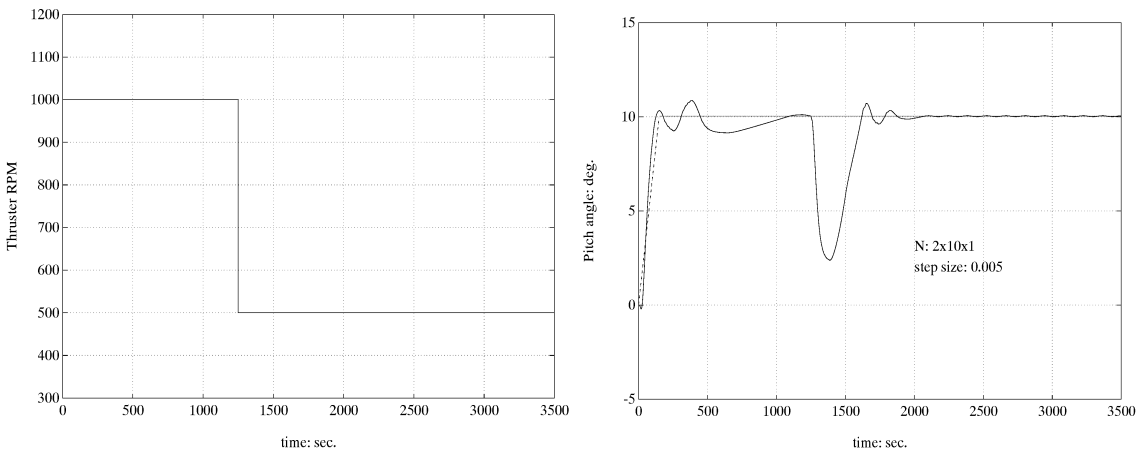



**Figure 5.9 Performance of the controller under sudden disturbance**
**(a) Change in forward thruster r. p. m.**
**(b) Response of the vehicle**

Again a sudden transition in command signal is found to create large oscillations in the depth because of the vehicle inertia and hence the command signal is varied in a linear fashion initially. As pointed out earlier, such sudden transitions in command signals are not practical. In our case, the forward thruster rpm is initially set to 550 and when the depth is brought to –200 feet the rpm is decreased to 225. This reduction in rpm is a necessary operational requirement, whenever there is a change in the depth command signal, because of the constraints due to the vehicle inertia. A step size of 0.001 is used for the neural network controller. The corresponding control input is shown in figure 7.8b.

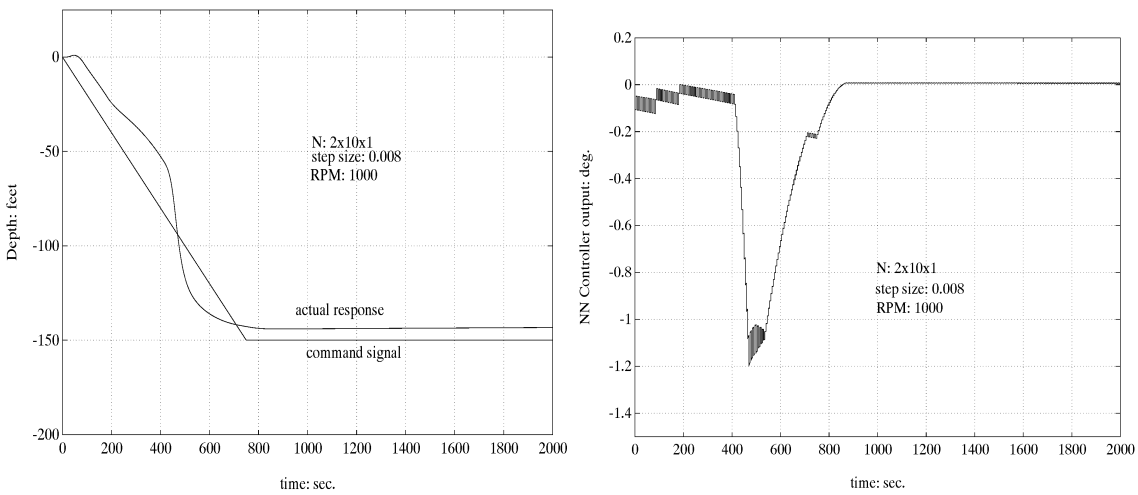



**Figure 5.8 Depth Control**

**(a) Response of the system for a commanded depth of 150 ft.**
**(b) Neural network output ( control input )**

### 7.3.4. Controller Performance under Disturbance:

The ability of the neural network controller to adapt to sudden changes in the dynamics is investigated by changing the vehicle speed suddenly and observing the tracking performance. Figure 7.9 shows the tracking of the pitch command when the vehicle speed was re-

ations are smaller than those of the static cases, a value of 0.001 is initially assigned. It is then adjusted to get a closer tracking and stability.

### 7.3.2. Experiment: 2.

Using the same control strategy, a neural network controller was designed for commanding the vehicle to track a specified heading(yaw). Figure 7.7 shows the response of the vehicle for a constant heading command of 30 degrees. Because of the inertia of the vehicle, the command signal was varied linearly during the initial period to avoid excessive overshoot. The corresponding control input to the rudder given by the neural network controller is shown in figure 7.4b. The network step size is 0.006 and the forward thruster rpm is 1000.

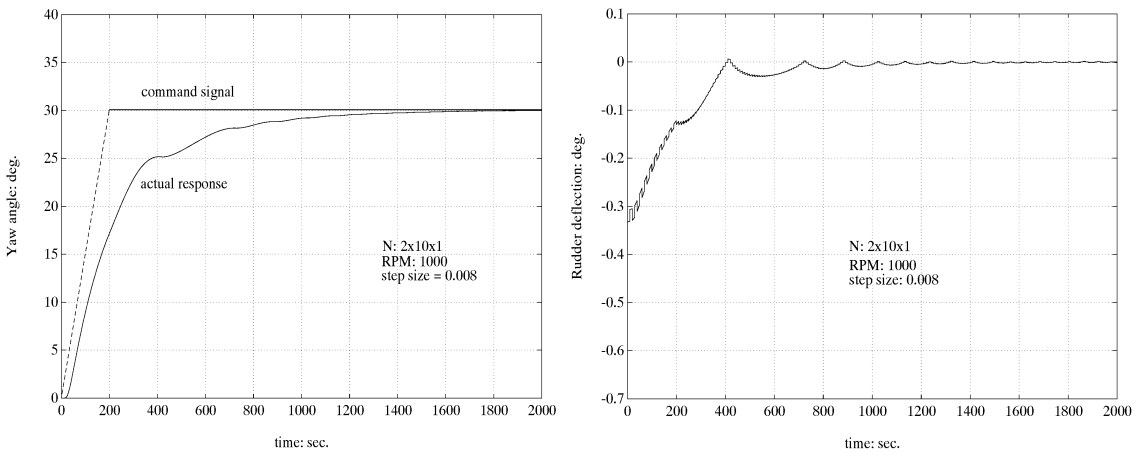



**Fig. 5.7 Heading control**

**(a) Vehicle response for a commanded heading of 30 degrees**
**(b) Corresponding control signal.**

### 7.3.3. Experiment: 3

Fig. 7.8 shows the response of the vehicle to a depth command signal which is meant to direct the vehicle to a depth of –200 feet in 1000 sec. and remain steady at that depth there after.

overshoot. From figure 7.5a, it can be seen that the vehicle is able to track the commanded pitch closely. The corresponding control input from the neural network is shown in figure 7.5b.

The tracking control of a time varying pitch command signal is shown in figure 7.6. The vehicle is commanded to be at a pitch of 10 degrees for the first 200 sec. and then at a pitch of –10 degrees for the next 1300 sec. The network is updated faster every 5 sec., whenever there is a change in the command signal. This proved to be a necessary requirement for closer tracking, based on empirical evidence. Figure 7.6b shows the corresponding control input generated by the neural network controller. The stepsize for the network updating is determined empirically. In some of the static pattern recognition simulations using the Alopex algorithm[58][60][61], it was observed that the stepsize may be of the order of 0.001 to 0.01, for proper convergence. Since the learning rates generally used in the case of control applic–

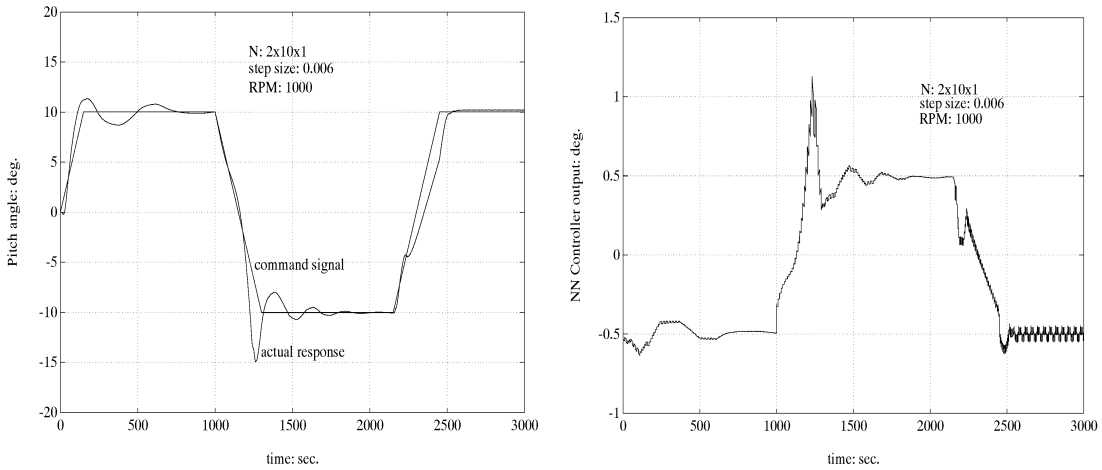



**Figure 5.6 Tracking of a time varying command signal ( pitch )**

**(a) Command signal and the vehicle response**
**(b) Control input to the dynamics**

$$E = \frac{1}{2T} \sum_{t=-T/2}^{T/2} \| y_d(t) - y(t) \|^2 \qquad (4.15)$$

The networks are updated at each $T$ sec. using the equations (3.12 to 3.16). A batch length of 10 sec. is used in the present studies. Whenever there is a sudden change in the error or when the command signals vary suddenly, the updating rate is reduced to 5 sec., to enable faster updating automatically..

## 7.3. RESULTS

### 7.3.1. Experiment: 1

The effectiveness of the controller in keeping the vehicle at a specified constant pitch is investigated in this case. The stern plane deflection is the control input to the dynamics. The response of the vehicle for a step pitch command of 10 degrees is shown in figure 7.5. The network is updated every 10 sec. using the total system error with a step size of 0.008 and a learning rate of 1.0. The command signal is varied linearly in the initial stage to reduce the

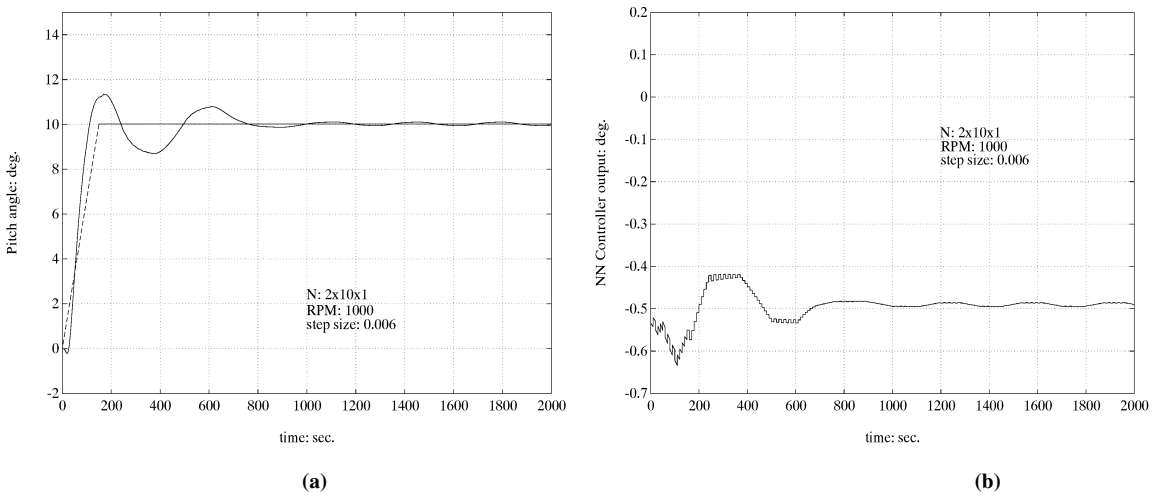



**Figure 5.5 Control of the Pitch angle**

**(a) Response of the vehicle for a commanded input of 10 degrees**
**(b) Control signal generated by the neural network controller**

delayed system response as the inputs. Since the Alopex algorithm does not need explicit error information at the output of the neural network, the updating of the weights can be done based on the total system error $e_y$. The complexities involved in using a direct control architecture with backpropagation algorithm are avoided by using the Alopex algorithm.

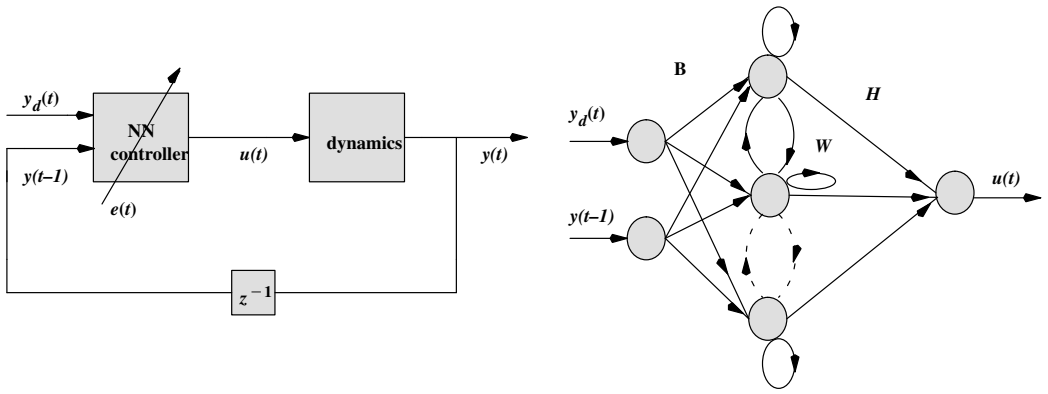



**Figure 5.4 Recurrent neural network based direct control scheme.**
**The network is adapted using the Alopex algorithm**
**(a) The control scheme**
**(b) The network architecture. The hidden layer is fully recurrent.**
**All weights are updated synchronously based on the system error**
**( $y_d - y$ )**

### 7.2.2. Batch Updating:

As the weight updating using the Alopex algorithm is done without explicitly calculating the error derivatives, it may be possible at some time instants to have the updating to be in the direction opposite to the error gradient. Hence, the strategy of instantaneous updating of the network at each sampling instant usually employed in the case of deterministic algorithms may not be desirable[62]. A *'batch updating'* over time is employed in the present cases in which the output error is accumulated over $T$ sec., to calculate the accumulated error $E$. That is,

ter 3. See [49][81][92][106][113] and [104] for details on similar recurrent network architectures used in applications involving dynamical systems.

It was earlier pointed out that recurrent architectures provide better response characteristics when identifying dynamical systems. One essential aspect which allows the recurrent networks to perform well is the stability of the equilibrium points[81][82]. The stability properties of the network equilibrium points and the convergence properties of the learning procedure, play equally important roles in the tracking performance, when neural networks are used as controllers. In recent years, a number of researchers have studied these properties on a variety of recurrent neural network architectures[81][82][131][132]. Symmetric, fully recurrent networks with no self feedback loops (generally called as Hopfield model[134][135]), are shown[7][8] possess stable equilibrium points for appropriately defined optimization functions. It is also shown that Hopfield model is a specific subclass of the networks represented by the equation (7.14) [131]. Further, it is shown that [131] the maximum number of stable equilibria is of the order $2^n$, where $n$ represents the number of neurons.

The above discussion on the equilibrium states for the networks represented by equation 6.1 was to point out that by an appropriate learning algorithm, such networks could be driven to a stable state. In the present study our aim is to apply the Alopex algorithm on the network architecture described above. It was already shown in Chapter 4 that the Alopex algorithm performs well on such on–line learning control applications, such as the motion control of the AUV.

### 7.2.1. Control Scheme

A direct control scheme similar to the one discussed in Chapter 4, is employed in the simulation studies. The studies are restricted to the SISO cases. The block schematics and the network architecture are shown in figure 7.4 The architecture uses the command signal and the

$u(k) = \sin(2\pi k/500)$, after training the network for 75, 000 iterations. The step sizes used for network weights, memory neuron weights and the memory coefficient are 0.005, 0.005 and 0.001, respectively. It can be seen that the neural network is able to identify the nonlinear dynamical system very well.

The identification results of the same dynamical system, but using a feedforward backpropagation neural network, is given in [51]. For that scheme all the five states, $y_p(k),\ y_p(k-1),\ y_p(k-2),\ u_p(k)\ \&\ u_p(k-1)$, are to be given as explicit inputs, and hence the network is more and the training duration is longer. Thus the memory neuron network approach provides considerable reduction in the number of network interconnections and training duration.

## 7.2. ALOPEX ALGORITHM FOR TRAINING RECURRENT NEURAL NETWORKS

In this section we present the results of the studies on the Alopex algorithm as a learning algorithm for recurrent networks. The neural network architecture used in the studies is of the form:

$$\begin{aligned} \dot{x} &= -x + W f_1(x) + B\ i(k) \\ y &= f_2(Hx) \end{aligned} \tag{4.14}$$

$x \in \Re^q$, $W \in \Re^{qxq}$, $B \in \Re^{qxp}$, and $H \in \Re^{rxq}$, where *p, q,* and *r* are the number of neurons in the input, hidden and output layers. The functions $f_1(.) : \Re^q \rightarrow \Re^q$, $f_2(.) : \Re^q \rightarrow \Re^r$ are vector valued sigmoidal transformation functions, with bounds between *–1.0* and *+1.0*. The corresponding network architecture is shown in figure 7.4b. The network has a feedforward input layer, a totally recurrent hidden layer and a feedforward output layer. Note that this architecture is same as the one used for the temporal XOR classification problem discussed in Chap-

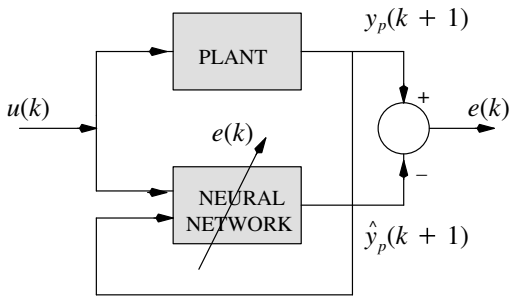



**Figure 5.2 Identification Scheme**

initial condition, the neural network weights are updated at each instant based on the error *e(k)*, using the equations (6.3 – 6.13). The identification is done using a random input signal *u(k)*, uniformly distributed in the interval [–1, 1], as the input. Figure 7.3 shows the output of the neural network and the nonlinear dynamical system for an input signal of the form,

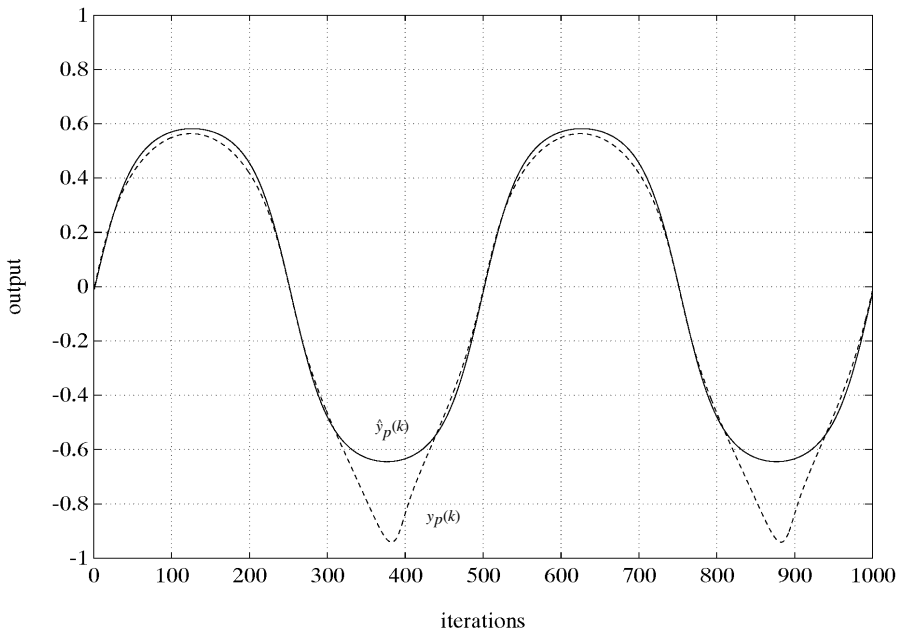



**Figure 5.3 Output of the plant and the identified model, after stopping the adaptation at 75, 000 iterations.**

$\Delta w_{ij}(n)$ , $\Delta x_{ij}(n)$ and $\Delta E(n)$ are the changes in the weights $w_{ij}$ , $x_{ij}$ and the error measure $E$ over the previous two iterations. As before, in the expressions (7.7) and (7.8), $T_w$ and $T_x$ are positive temperatures that determines the effective randomness in the system. The simulations are started with a large value for the temperatures. Subsequently, the temperatures are set equal to the average value of the corresponding correlation measure.

The memory coefficients $\alpha_j$ are updated using a simple correlation measure;

$$\alpha_j(n) = \alpha_j(n-1) + \delta\alpha_i \qquad (4.11)$$

$$\delta\alpha_j(n) = \begin{cases} -\delta_\alpha & \text{if } \Delta\alpha_j.\Delta E \geq 0 \\ +\delta_\alpha & \text{if } \Delta\alpha_j.\Delta E < 0 \end{cases} \qquad (4.12)$$

where $\delta_\alpha$ is a small positive step–size for the memory coefficient.

### 7.1.3. Results

A system identification experiment is designed to evaluate the memory neural network trained using the Alopex algorithm and results are presented. The objective is to identify a nonlinear dynamical system of the form[51]:

$$y_p(k+1) = \frac{y_p(k)\ y_p(k-1)\ y_p(k-2)\ [\ y_p(k-2) - 1\ ]\ u(k-1) + u(k)}{1 + y^2{}_p(k-3) + y^2{}_p(k-2)} \qquad (4.13)$$

Note that the neural network has to retain the previous states of the inputs and outputs for a proper identification of the system and hence a feed forward network will not be able to map the system unless all the required states are explicitly presented as inputs. The identification scheme is shown in figure 7.2 A network of size 2x10x10x1 is used to model the dynamical system. Signals $u(k)$ and $y_p(k)$ form the input to the network. Starting from a random

rons ), $x_{ij}$ ( between the memory neurons and network neurons ), and the memory coefficient $\alpha_j$ ( see Section 6.3.1 for details on the notations ).

The interconnection strengths are modified similar to the learning scheme used in the static learning tasks using the Alopex algorithm ( discussed in Chapter 3 ). The weights are modified by a constant step–size δ, based on a probability measure. The procedure is outlined below:

During the n$^{th}$ iteration, the weight $w_{ij}$ is updated as,

$$w_{ij}(n) = w_{ij}(n-1) + \delta w_{ij}(n) \qquad (4.3)$$

$$x_{ij}(n) = x_{ij}(n-1) + \delta x_{ij}(n) \qquad (4.4)$$

where,

$$\delta w_{ij}(n) = \begin{cases} -\delta_w & with\ probability\ \ P^w{}_{ij}(n) \\ +\delta_w & with\ probability\ \ (1 - P^w{}_{ij}(n)) \end{cases} \qquad (4.5)$$

$$\delta x_{ij}(n) = \begin{cases} -\delta_x & with\ probability\ \ P^x{}_{ij}(n) \\ +\delta_x & with\ probability\ \ (1 - P^x{}_{ij}(n)) \end{cases} \qquad (4.6)$$

The probabilities $P^w{}_{ij}(n)$ & $P^x{}_{ij}(n)$ as in equation(3.7) are the Boltzmann distribution given by:

$$P^w{}_{ij}(n) = \frac{1}{1 + exp\ (-\Delta^w{}_{ij}(n)/\ T_w)} \qquad (4.7)$$

$$P^x{}_{ij}(n) = \frac{1}{1 + exp\ (-\Delta^x{}_{ij}(n)/\ T_x)} \qquad (4.8)$$

where $\Delta^w{}_{ij}(n)$ and $\Delta^x{}_{ij}(n)$ are given by the correlation:

$$\Delta^w{}_{ij}(n) = \Delta w_{ij}(n).\Delta E(n) \qquad (4.9)$$

$$\Delta^x{}_{ij}(n) = \Delta x_{ij}(n).\Delta E(n) \qquad (4.10)$$

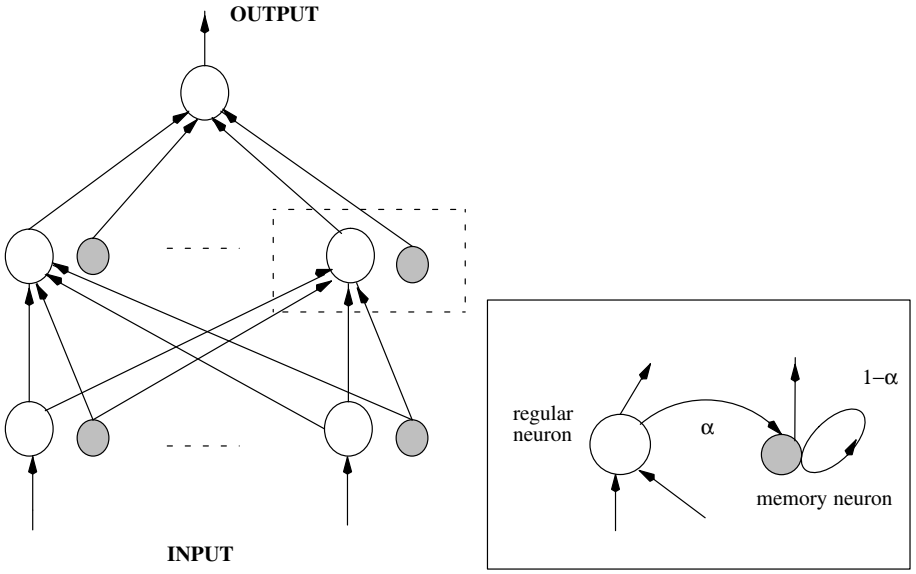



**Figure 5.1** **Memory neuron network architecture. The details of the information transfer between the network neuron and memory neuron is shown in the inset.**

### 7.1.2. Learning

The error function to be minimized is chosen as the squared error at the output of the network,.

$$e_p = \frac{1}{2}\sum_{i=1}^{N}(t_i - o_i)^2 \tag{4.1}$$

where $t_i$ and $o_i$ are the target and actual output, respectively, for the neuron $i$ in the output layer ( consisting of $N$ neurons ) for sample $p$. In the case of a pattern matching problem, the error $e_p$ could be summed over the entire length of the sequence.

$$E = \sum_{p} e_p \tag{4.2}$$

The learning process involves the modification of the weights $w_{ij}$ ( between the network neu-

this scheme. It is also shown that a Alopex–trained neural network has better measurement noise rejection capabilities than the backpropagation–trained network over a large range of signal to noise ratio.

## 7.1. ALOPEX ALGORITHM FOR TRAINING MEMORY NEURON NETWORKS

### 7.1.1. Memory Neuron Network Architecture:

A brief discussion on the memory neuron network architectures was provided in the previous chapter. Basically, such networks are simple approaches for introducing memory into the network, while eliminating many of the limitations of the *'parallelizing the time'*[92][123] approaches and the *'time delay'* approaches[48][72]. The stability problems and computational complexities associated with the recurrent networks are thus alleviated. Both of the above architectures use backpropagation as the learning algorithm, with necessary modifications.

The objective here is to investigate the effectiveness of the Alopex algorithm on the memory neural network architecture[105] to solve dynamical problems. The architecture employed is shown in figure 7.1 Each memory neuron has an input connection of strength $\alpha$, (the memory coefficient) from the corresponding network neuron and a self feedback connection of strength $(1 - \alpha)$; both inputs have a time–delay of one cycle time. The only input connections to the memory neurons are from its network neuron, while outputs of the memory neurons are connected to all the network neurons in the upper layer. The self feedback connection enables the memory neuron to retain the information contents of the previous time–steps. The network dynamics remains the same as the one discussed in Section 6.3.1. To incorporate input signals of range [–1, 1], a sigmoid transformation of range [–1, 1] is employed.

# CHAPTER 7.

# LEARNING IN RECURRENT NEURAL NETWORKS USING THE ALOPEX ALGORITHM

In the previous chapter, different recurrent network architectures and learning algorithms for applications involving dynamical systems were reviewed. These architectures and algorithms differ in their performance aspects in terms of computational complexity, mapping capabilities, convergence and stability. Hence, choosing an appropriate architecture and the learning algorithm is much problem dependent. In Chapter 3, it was pointed out that the Alopex algorithm could be used for training neural networks of any architecture, including those having asymmetric feedback connections. The studies on the classification of temporal XOR patterns were provided to demonstrate the ability of the algorithm. In this chapter those studies are extended to learning control applications. Also, the Alopex algorithm is introduced as a learning algorithm for the memory neuron networks[105][110]. For the autonomous underwater vehicle control application, it is shown that the recurrent neural networks trained using the Alopex algorithm gives good tracking performance and stability[62]. Further, it is demonstrated that specific improvements in the network complexity can be achieved using

tion for the neural networks concerning the on–line learning control of autonomous underwater vehicles is portrayed. A number of other applications for neural networks are also introduced in this dissertation. Specifically, they refer to the continuous recognition of undersea targets from sonar returns and the handwritten digit recognition using the Fourier Descriptors.

characteristics of the vehicle. Such a scheme needs prior information about the vehicle characteristics for the design of the fuzzy controller. Starting the controller from random initial conditions is the main cause for the high initial overshoot in the simulation results discussed in Chapter 5. Also, the error feedback scheme suggested in Section 4.4.7. is not employed in those studies. Employing a pre–trained controller and introducing the error feedback in the control scheme are two possible extensions.

## 3.3. CONCLUSION

The Alopex algorithm is introduced as a universal learning algorithm for connectionist models. It is shown that the algorithm has a number of properties which could be used effectively in situations where traditional gradient descent based algorithms are ill–suited. The universality of the algorithm is demonstrated by studying it on a number of different network architectures. The algorithm is very simple. The computational complexity of the algorithm does not depend up on the network architecture. We have demonstrated this property on recurrent neural network architecture with asymmetric feedback connections for complex learning tasks such as the control of an underwater vehicle. The learning, generalization and temporal information extraction abilities of the algorithm are studied on a variety of problems and the results confirm that the Alopex can be used as an efficient learning algorithm. The Alopex algorithm is found to possess better generalization properties than other learning algorithms. Also it is worth noting that only a basic version of the Alopex algorithm is used in the studies. A variety of modifications of the algorithm is possible which can effectively enhance many required properties.

Two new schemes for the direct on–line learning control of dynamical systems has been introduced. The proposed schemes give better tracking performance and dynamic response characteristics than the other existing methods. Using the proposed scheme a new applica-

*(d) Gain Layer schemes and extensions to MIMO systems:*

This research has provided two improved direct control schemes for the on–line learning control of dynamical systems. Detailed studies were restricted to SISO linear and nonlinear dynamics. We demonstrated the effectiveness of such an approach for simple MIMO linear dynamics. These studies could be extended to general nonlinear MIMO dynamical systems.

*(e) Improving the dynamic response: Adapting the shape of the velocity reference signal.*

The velocity reference scheme discussed in this dissertation incorporated a carefully picked reference signal. In practical situations, selecting the characteristics of the reference signal may be difficult. An on–line adaptation of the characteristics of the of the reference signal can make the selection easy and improve the tracking performance .

*(f) Continuous recognition of sonar targets:*

The sonar target recognition scheme described in Chapter 3., takes a fixed number of discrete return signals at a time, for the classification. A proper modification in the network architecture would enable the recognition of continuous signals, with no need of separation of individual returns.

*(g) Dynamic response of the controllers for the AUV:*

It is to be stressed here that, for the control schemes discussed in this study, the neural network controllers start training from random initial weight distribution. No prior training is involved. This approach differs from the method described by Fujii and Ura[138] for the control of AUVs. They employed a *'premature'* fuzzy controller to take care of the vehicle motion till the neural network controller learns the dynamic

NMDA receptors present in the sensory pathway is capable of carrying out temporal correlations between the inputs[45]. Hence correlation based learning algorithms such as the Alopex are capable of being carried out by existing circuitry of the brain. There is also considerable randomness in the responses of neurons[136] and any learning algorithm that is used for modelling the brain functions should consider this property.

## 3.2. SCOPE FOR FUTURE WORK

The research has provided foundation for a number of possible theoretical as well as experimental extensions. They include;

*(a) Convergence proof for the Alopex algorithm:*

In this dissertation, even though a wide variety of network architectures and problems are investigated using the Alopex algorithm, the studies are restricted to experimental analysis. The proof of convergence for Alopex algorithm as a learning algorithm for neural networks is a relevant research area.

*(b) Optimum cost function for specific classes of problems:*

It is shown in this study that selecting a proper cost function, the speed of convergence and the classification accuracy can be improved. For specific classes of problems smoother error surface and hence fewer number of local minima can be obtained by the proper choice of a cost function. Further study is required in categorizing the learning tasks and proposing their optimum cost functions

*(c) Alopex as a learning algorithm for reinforcement learning tasks:*

The studies described in this dissertation is restricted to the application of the Alopex algorithm as a supervised learning algorithm. Extension of the algorithm as a reinforcement learning algorithm is possible. Note that there is considerable interest in reinforcement learning in recent times, especially for control applications[137].

shown that an additional reference provided for the error change signal, improves the tracking performance of the system. Also, the velocity reference scheme can be combined with the gain layer schemes to get further improvement in the response.

Extensive studies are conducted to investigate the effect of the learning rate on the dynamic response, tracking performance and the ability to respond to disturbances, when neural networks are used for on–line learning control applications. The learning rate is found to be a critical factor for the overall stability of the system. It is the single most important parameter of the network which affects the performance of the systems. The possible learning rates for the control applications are found to be much smaller than the rates used for static learning. These results agree with the theoretical studies reported by Yabuta *et al.*[101]. For the control of a special class of MIMO dynamical systems where the controller can be split into single–input single–output networks, different learning rates are needed for networks to perform optimally. This indicates that the control of MIMO systems using a single neural network, with a single learning rate, may not give an optimum performance.

The studies on the adaptability of the controllers to parameter disturbance, again indicate the importance of choosing a proper learning rate. A higher learning rate produces larger initial overshoot, but it makes the controller easily adaptable to changes in the dynamics. It is found that with a proper learning rate, the network is able to adapt well in presence of sudden and slow varying parameter disturbances.

It was indicated in the previous chapters that Alopex is more plausible biologically than many of the other learning algorithms such as backpropagation. As Hinton[31] and Baldi[34] point out, backpropagation is not biologically plausible. There is no evidence that the synapses in nervous systems could be used in backward direction or neurons can propagate error derivatives in the reverse direction. On the other hand, there is considerable evidence that nervous systems carry out correlation between the inputs[1][34]. For example, the

The studies on static and temporal sequence recognition tasks are then extended to the learning control tasks. Using a feedforward network architecture, the Alopex algorithm is shown to be able to track the control commands well. The studies are conducted on the fifth order linearized dynamical model of an underwater vehicle. One important advantage of using Alopex for control applications is that an exact error gradient is not required and the algorithm can be readily used for direct control applications. On the other hand, since the updating is done based on a probability measure, on–line learning produces a constant steady state error. One way to reduce this error is to adapt the step size when the command signal vary slowly. Another important observation on the algorithm is that the instantaneous updating of weights is not desirable for Alopex based controllers. The updating is to be based on an accumulated error over a small period of time.

The universality of the Algorithm is further investigated by analysing a recurrent network controller trained using the Alopex algorithm for the on–line control of the nonlinear vehicle dynamics. The results show that the algorithm gives equally good tracking performance and stability as the learning controllers designed based on the gain–layer schemes trained using the backpropagation algorithm. The comparison of the backpropagation and the Alopex performance over a wide range of noisy data show that Alopex has better noise rejection capabilities than backpropagation.

Two direct control approaches for the learning control using gradient based neural network approaches are proposed, in this dissertation. The proposed schemes (gain layer schemes) overcome some limitations of the existing approaches and are shown to provide better tracking performance. The gain layer schemes are further tested rigorously as the learning controllers for autonomous underwater vehicles. Studies are conducted on linearized model as well as nonlinear model of the vehicle. A scheme for improving the dynamical response of the neural network controllers using velocity reference feedback also is discussed. It is

tract temporal information from consecutive input patterns. This property is further studied on the temporal XOR problem using a recurrent neural network architecture. The network is able to correctly classify temporal XOR sequences of delays 0, 1, and 2 using the Alopex algorithm. The above results also emphasize the universality of the Alopex algorithm.

**Table 8.1. Performance Comparison of Different Learning Algorithms on MONKS Problems[88]**

| ALGORITHM | PROBLEM #1 | PROBLEM #2 | PROBLEM #3 |
|---|---|---|---|
| AQ17−DCI | **100%** | **100%** | 94.2% |
| AQ17−HCI | **100%** | 93.1% | **100%** |
| AQ17−FCLS | | 92.6% | 97.2% |
| AQ14−NT | | | **100%** |
| AQ15−GA | **100%** | 86.8% | **100%** |
| ASSISTANT PROFESSIONAL | **100%** | 81.3% | **100%** |
| mFOIL | **100%** | 69.2% | **100%** |
| ID5R | 81.7% | 61.8% | |
| IDL | 97.2% | 66.2% | |
| ID5R−hat | 90.3% | 65.7% | |
| TDIDT | 75.7% | 66.7% | |
| ID3 | 98.6% | 67.9% | 94.4% |
| ID3, no windowing | 83.2% | 69.1% | 95.6% |
| ID5R | 79.7% | 69.2% | 95.2% |
| AQR | 95.9% | 79.7% | 87.0% |
| CN2 | **100%** | 69.0% | 89.1% |
| CLASSWEB 0.10 | 71.8% | 64.8% | 80.8% |
| CLASSWEB 0.15 | 65.7% | 61.6% | 85.4% |
| CLASSWEB 0.20 | 63.0% | 57.2% | 75.2% |
| PRISM | 86.3% | 72.7% | 90.3% |
| ECOBWEB leaf prediction | 71.8% | 67.4% | 68.2% |
| ECOWEB l. p. & information utility | 82.7% | 71.3% | 68.0% |
| BACKPROPAGATION | **100%** | **100%** | 93.1% |
| BACKPROPAGATION with weight decay | **100%** | **100%** | 97.2% |
| CASCADE CORRELATION | **100%** | **100%** | 97.2% |
| ALOPEX | **100%** | **100%** | **100%** |

On the standard benchmark problems, the Alopex algorithm is found to need comparable number of iterations as the backpropagation. Since the concept of iteration is not same for the learning algorithms, it may not be the best way to evaluate relative merits of an algorithm. In the Alopex learning, all the weights are updated synchronously and hence in a parallel hardware implementation, all the interconnections can be updated in one clock cycle. This is valid even when the algorithm is used with any type of architecture, including the feedback configuration. This property of Alopex is very relevant in hardware implementation of dynamical networks.

It is to be emphasized here that for static pattern classification tasks, the generalization capabilities of the leaning algorithm may be more important than the training time. Generalization refers to the ability of the trained networks to corectly interpret a new set of input data from the same domain of the training data. It is the capability of the networks to learn the regularities in the input data rather than performing a *'look–up table'* operation.

The generalization properties of the Alopex algorithm is investigated using the set of MONKS problems. The training set in each of these problems is smaller than the test set and the underlying representation function in each case is different. From the results of the important machine learning and neural network approaches[88] for the MONKS problems (presented in table 8.1), it is interesting to note that many of the algorithms fail to generalize when the input data is noisy. The symbolic learning algorithms such as AQ, are able to classify 100% of the test data correctly, under such conditions, though they failed in the some of the other sets of problems. The results using Alopex higlight the generalization ability of the algorithm and are the best reported results so far on the MONKS problems.

The practical applicability of the Alopex algorithm for classification is demonstrated on two complex tasks such as sonar target recognition and handwritten digit recognition. The sonar target recognition experiment also gives an indication of the ability of the algorithm to ex-

# CHAPTER 3.

# SUMMARY AND CONCLUSIONS

## 3.1. SUMMARY OF RESULTS

Through a number of experimental studies, it is shown that the Alopex algorithm can be used as an efficient learning algorithm for connectionist models. The problems studied include a number of standard benchmark problems such as XOR/Parity, Symmetry and Encoder as well as, a variety of complex application–specific problems in classification, modelling and learning control. For the benchmark problems, studies are conducted using three different error measures to investigate the impact of an appropriate cost function for learning. The Alopex algorithm provides a stochastical approach for the connectionist learning. Such an approach overcomes a number of limitations of the traditional gradient decent based learning algorithms, viz., such as convergence to local minima, sensitivity to initial weight distribution and necessity of a monotonically increasing, differentiable transfer function.