may be noted that the proposed schemes for direct control of dynamical systems, are not restricted to the applications using backpropagation. It could be used with any learning algorithm which requires the explicit error information at the output of the network.

A scheme for improving the dynamic response of the neural network controllers using velocity reference feedback was also discussed. This is achieved by providing an additional reference signal for the error derivative. Also, the velocity reference scheme can be combined to the gain layer schemes to consolidate the improvement in the response.

Figure 4.16 provides the control signals generated by the neural network for the three cases referred in Fig. 4.13. It can be verified that with the velocity reference scheme, the network is able to generate a control signal which results smaller values of rise time, overshoot and settling time.

2.6. SUMMARY

In this chapter, the temporal pattern recognition studies using the Alopex algorithm is extended to learning control applications. It is demonstrated that the Alopex can be used effectively for learning control applications. Some of the important observations made during this study are:

(a) Instantaneous updating of weights is not a desirable way of training controllers using the Alopex algorithm. The updating should be based on an accumulated error over a small period of time.

(b) Because of the stochastic nature of the algorithm, a perfect tracking of the commanded signal may be difficult, if the stepsize is constant. One way to approach this problem is to progressively reduce the stepsize as the network converges. This idea will be discussed in detail in the subsequent chapters.

The present chapter also dealt with the difficulties in using backpropagation for direct control applications. The existing methods for direct control applications using backpropagation algorithm were reviewed and analyzed. Two modifications for the existing schemes were proposed. The presented methods avoid the need for error transformations required in other schemes and improve the system response using error feedback. Some examples are provided to verify the superior tracking and dynamical responses of the proposed schemes. It

Representing the system trajectory in a state space form (figure 4.14), it can be seen that the large rise time is due to the fact that the initial control input by the network is not accelerating fast enough. Also the network is not able to decelerate the control input at the switching point e=0, for a lesser overshoot. These shortcomings can be blamed on the inadequacy of information presented to the network. Simulation studies also verified that an increase in number of layers or number of interconnections, does not improve the system performance[64][67].

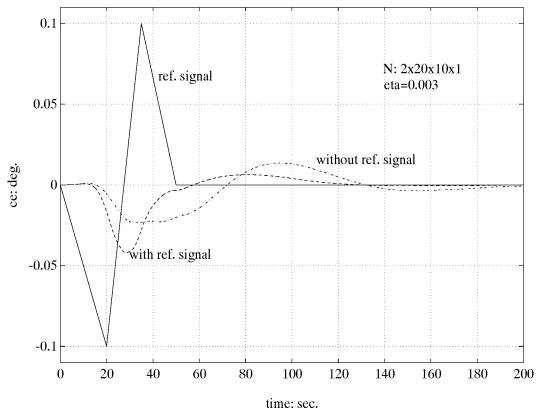

Figure 3.15 The velocity reference signal and its effect on the change in $\dot{e}$

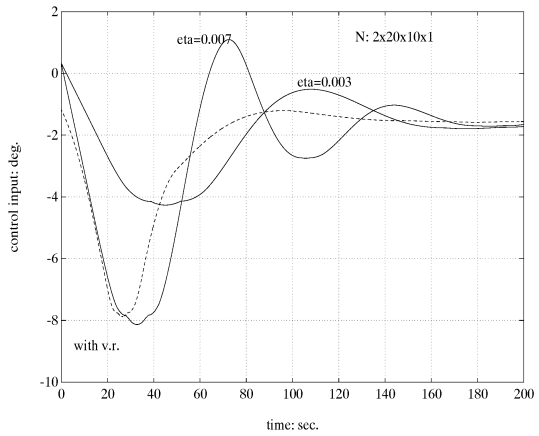

Figure 3.16 The control signals generated by the neural network

Adding the reference signal for $\dot{e}_i$ (figure 4.15.) as the second input signal to the network, forces the system trajectory to track in a better way. The reference signal helps the network to drive the system faster in the initial region region of the state space trajectory $(+e, -\dot{e})$, and decelerate faster in the $(-e, -\dot{e})$ region, similar to a classical Proportional Derivative (PD) controller. Figure 4.13 compares the response with and without the velocity reference feedback. With the velocity reference feedback the system rise time is improved to 27 sec., the overshoot and the settling time are reduced to 42% and 150 sec. respectively.

2.5.2. Simulation Results:

The proposed scheme was applied on the same nonlinear underwater vehicle dynamics described before (equation 4.47). The response of the vehicle for a commanded pitch of 10 degrees is shown in figure 3.13. The neural network controller is of size 2x20x10x1, having two hidden layers of 20 and 10 neurons each. For a learning rate of 0.003, the rise time for the system is about 42 sec. with an overshoot of 70%. The time taken by the network to settle to the commanded pitch is more than 200 sec. The learning rate determines the time constant of the systems and is the most important parameter affecting the transient response. For example,

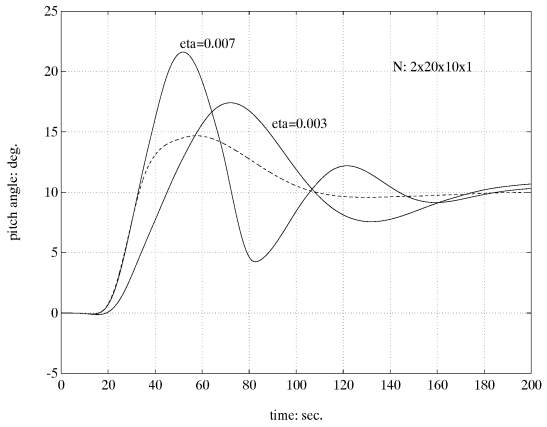

Figure 3.13 Vehicle response for a commanded pitch of 10 degrees The solid lines represent the response with velocity reference.

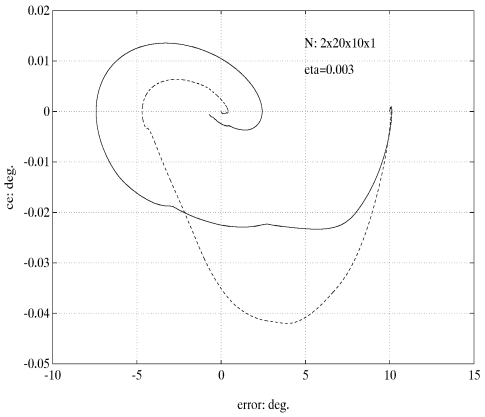

Figure 3.14 System trajectory for the response shown in Figure 4.13.

the rise time could be improved by increasing the learning rate. For a learning rate of 0.007, the rise time is smaller than that for 0.003 but the overshoot is more than 100%. Obviously, the response is more oscillatory and settling time is more than 200 sec.

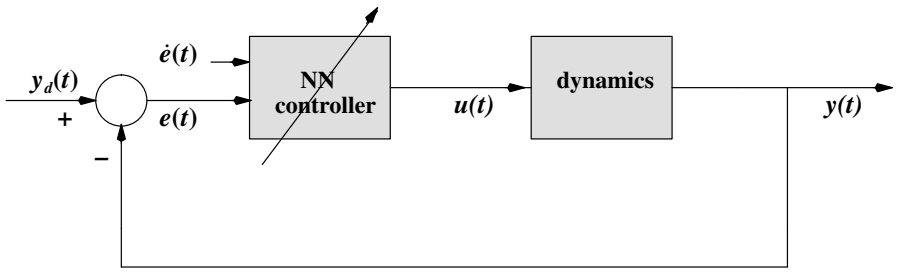

Figure 3.12 Controller architecture using neural network.

$$E = \frac{1}{2}A_1 e_i^{\,2} + \frac{1}{2}A_2 \dot{e}_i^{\,2} \tag{4.48}$$

where e_i and $\dot{e}_i$ are the error and change in error given in equations (4.49) and (4.50). The parameters A_1 and A_2 are constant scaling factors for e_i and $\dot{e}_i$ respectively and are chosen empirically so that the contributions from both the errors are approximately the same. Expressing the errors as a function of time,

$$e_i(t) = y_d(t) - y(t) \tag{4.49}$$

$$\dot{e}(t) = e(t-1) - e(t-2) \tag{4.50}$$

Hence the network updating equations (4.29) and (4.13) become:

$$\delta_j^{\,p} = (A_1 e_i + A_2 \dot{e}_i) f'(net_j^{\,p}) \qquad \text{for output layer} \tag{4.51}$$

and $$\delta_i^{\,p} = f'_i \sum_i \delta_j^{\,p} w_{ji} \qquad \text{for any other layers}$$

The network weights are first initialized to random values and the updating is done at each instant in time, according to the above set of equations. The error e_i is kept zero for the first iteration and $\dot{e}_i$ is kept zero for the first two iterations.

2.5. IMPROVING THE DYNAMIC RESPONSE USING VELOCITY REFERENCE FEEDBACK

In previous section, it was shown how an error feedback could be used to improve the dynamic response of the overall control system (figure 4.8). The networks considered had only one input viz., the desired system output y_d (or command signal). Experimental studies[53][66] have shown that an equivalent system performance could be obtained by replacing the input signal y_d by the error signal $(y_d - y)$. Figure 4.12 shows such a scheme. An important property of this control scheme as well as the other schemes discussed in this chapter is that the neural network weights are updated solely based on the error between the desired and actual response of the system. However it can be shown[64][67] that a neural network controller designed solely to realize a maximum acceptable steady state error, may have unsatisfactory transient response such as excessive overshoot and large rise time. Conversely, a controller designed to realize very good transient response may have unacceptable steady state error characteristic. This section investigates the effectiveness of adding a velocity reference feedback for simultaneously improving the transient as well as steady state error response of a neural network controller[64][67]. The system performance is monitored, both by the error as well as the change in error. In addition to the usual reference signal for the error(command signal) a second reference signal for the change in error is also introduced. It is shown that specific improvements in overshoot, rise time and settling time could be achieved using this method.

2.5.1. The Velocity Reference Scheme:

The control scheme employs two input signals, the error and change in error (see figure 4.12). The error function to be optimized is defined as

the on–line updating and the error feedback, provides better transient response and smaller tracking error. The variation of the gain factor *g(k)* as a function of time is shown in figure 4.10. Figure 4.11. demonstrates the effectiveness of the gain layer scheme for fast varying command signals. A command signal of the form

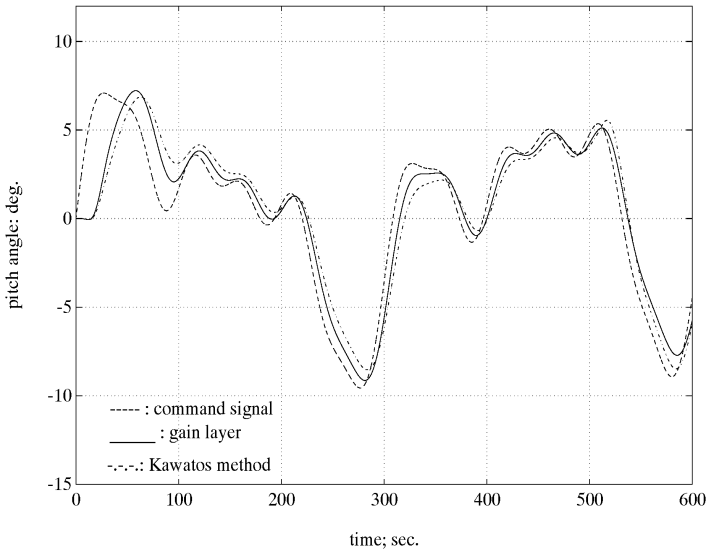

Figure 3.11 Comparison of Kawato's scheme and the 'gain layer' scheme for a time varying command signal.

$y_d(k) = 4\sin(\frac{2\pi k}{7000}) + 2\sin(\frac{2\pi k}{2000}) + 0.8\sin(\frac{2\pi k}{1000}) + 2.8\sin(\frac{2\pi k}{3000})$, is given as the input to the network. The results shown make use of a network of size 2x20x10x1 trained with a learning rate of 0.008. The controller is able to make the system track the command signal closely. The dotted curve corresponds to the gain feedback scheme with the error feedback. The gain K_k as well as the initial value of *g(k)* are kept the same as in the previous simulation. As in the previous example, the on–line adaptation in the gain layer scheme, drives the system towards better tracking.

The scheme which uses transformation of the error by backpropagating it through the dynamics, is found to be slow in tracking the command signal. The shown results are the best ones for each method over a number of trials with different parameters. The inverse transfer matrix scheme also suffer from poor tracking performance as the high values of the numerically computed inverse differential gain results in large fluctuations in the transformed error. In both cases, the learning rate used was 0.0001, and any attempt to increase it creates unstable responses. The feedback gain scheme and the gain layer schemes are found to provide good tracking performances.

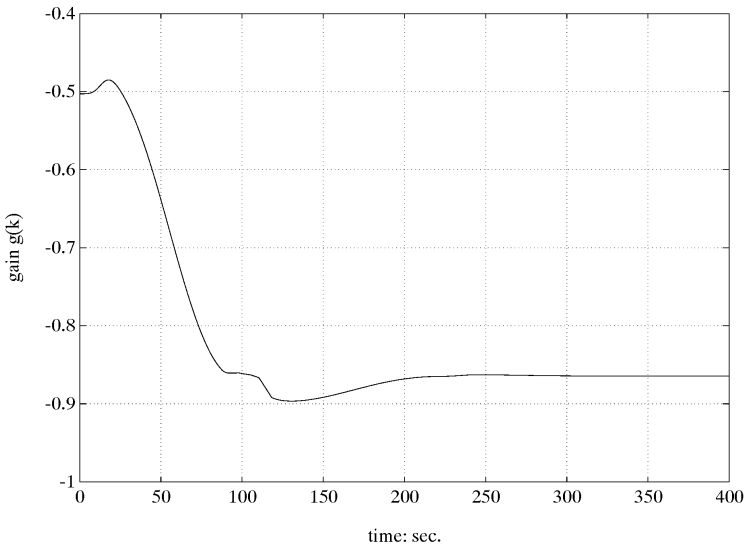

Figure 3.10 Variation of the gain factor for a response as shown in Figure 4.9.

The gain layer scheme with no error feedback performs significantly better than the 'specialized learning scheme' and the 'inverse transfer matrix scheme'. The gain K_k (see figure 4.5) for the feedback gain scheme is found by linearizing the vehicle dynamics about the operating point and estimating the inverse d.c. gain. The gain layer g (figures 4.7 and 4.8) is initialized to the same value and updated on–line. It can be seen that the gain layer scheme using

pitch. Assuming that the vehicle is travelling at a constant speed, the dynamics for the above relationship could be expressed by the following nonlinear equation[95].

$$\sum M_M = I_y\dot{q} + (I_x - I_y)rp - (\dot{p} + qr)I_{xy} + (p^2 - r^2)I_{xz} + (qp - \dot{r})I_{yz} + m[z_G(\dot{u} - vr + wq) - x_G(\dot{w} - uq + vp)] \quad (4.47)$$

where $\sum M_M$ corresponds to the summation of all the moments with respect to the *y* axis, which is a nonlinear function involving the 'stern plane' deflection δs. *u,v* and *w* are the linear velocities along the three co–ordinate axis and *p, q,* and *r,* the angular velocities. *'I'* stands for the moment of inertia, with the subscripts indicating whether planar or axial and *m,* the mass of the vehicle (details on the equations of motion is given in Appendix C). The normal operating range of the vehicle pitch is +15 degrees to –15 degrees. Figure 4.9 compares the results with different schemes for a commanded pitch of 8.5 degrees using a network of size 2x20x10x1.

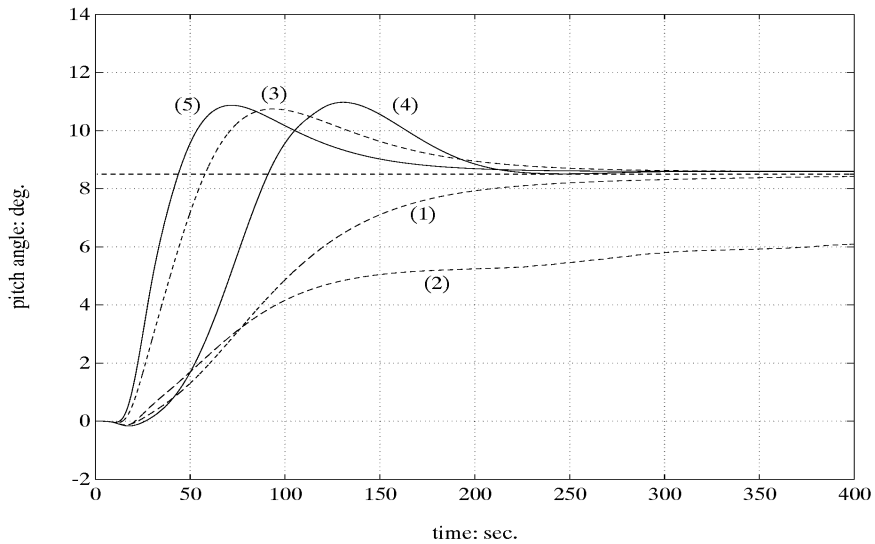

Figure 3.9 System response for a commanded pitch of 8.5 degree.

(1) Specialized learning scheme
(2) Inverse Transfer matrix scheme
(3) Feedback gain scheme
(4) Gain layer scheme (without error feedback)
(5) Gain layer scheme (with error feedback)

In this case, the equation (4.37) gets modified as,

$$g(k) = g(k-1) + \alpha e_y J(u) \frac{[v + e_y]}{1 + gJ} \tag{4.46}$$

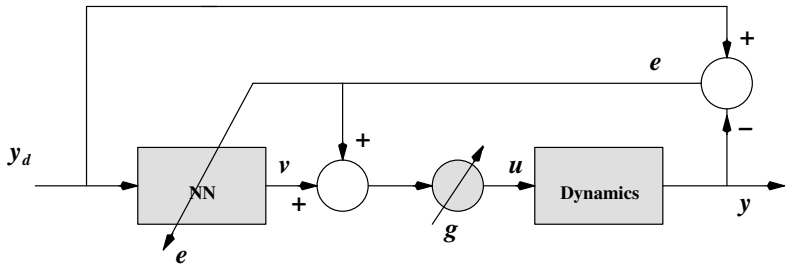

Figure 3.8 The Gain Layer Scheme with error feedback

The important differences between the gain layer scheme presented here and the error transformation schemes[52][[97–99] are: (i) We use the total system error than a transformed error in the case of other schemes. This reduces the fluctuations in the weight space due to the instantaneous computation of Jacobian and use it for network updating[54]. In the present scheme, the Jacobian does not directly affect the error based on which the networks are updated. It affects only the gain factor and the updating of the gain factor can be controlled by a different learning rate; it is found that this improves the performance. (ii) Because of the presence of error feedback, the system response is faster. The network time constant plays only a secondary role in the dynamical response of the system.

2.4.8. Simulation Results

We applied the proposed gain layer scheme for the control of a nonlinear underwater vehicle dynamics (discussed in Appendix C). The objective of the neural network controller is to give a correct deflection angle to the vehicle 'stern plane' actuator to maintain a commanded

Then, the control input to the dynamics is given by

$$u = zg \tag{4.36}$$

As before, defining

$$y = J(u)u \tag{4.37}$$

$$= J(u)g[v + (y_d - y)] \tag{4.38}$$

Expanding and rewriting:

$$y = \frac{J(u)vg}{1 + J(u)g} + \frac{J(u)y_d g}{1 + J(u)g} \tag{4.39}$$

Under steady state conditions, we could approximate $J(u) = \frac{1}{g}$, resulting in

$$y = \frac{v + y_d}{2} \tag{4.40}$$

Considering the updating equations of the gain factor g, we have,

$$\frac{\partial E_y}{\partial g} = \frac{\partial E_y}{\partial y}\frac{\partial y}{\partial g} \tag{4.41}$$

But

$$\frac{\partial y}{\partial g} = \frac{\partial y}{\partial u}\frac{\partial (zg)}{\partial g} \tag{4.42}$$

As $\frac{\partial (zg)}{\partial g} = z + g\frac{\partial z}{\partial g}$ and $\frac{\partial z}{\partial g} = -\frac{\partial y}{\partial g}$,

$$\frac{\partial y}{\partial g} = -e_y\frac{\partial y}{\partial u}[z + (-\frac{\partial y}{\partial g})g] \tag{4.43}$$

$$= \frac{\frac{\partial y}{\partial u}z}{(1 + g\frac{\partial y}{\partial u})} \tag{4.44}$$

Or

$$\frac{\partial E_y}{\partial g} = -\frac{e_y J[v + e_y]}{1 + gJ} \tag{4.45}$$

where $\Delta g(k)$ is the change in $g(k)$ at instant k. Using the gradient descent rule to update $g(k)$ to minimize the squared error E_y (equation 4.7), we obtain

$$\Delta g(k) = -\alpha \frac{\partial E_y(k)}{\partial g(k)} \tag{4.31}$$

where α is the updating rate for $g(k)$. Rewriting the above equation using the chain rule and dropping the indices for brevity,

$$\frac{\partial E_y}{\partial g} = \frac{\partial E_y}{\partial u}\frac{\partial u}{\partial g} \tag{4.32}$$

$$\frac{\partial E_y}{\partial u} = -(y_d - y)\frac{\partial y}{\partial u} \tag{4.33}$$

Since $\frac{\partial u}{\partial g} = v$, equation (4.30) reduces to:

$$g(k) = g(k-1) + \alpha(y_d - y)v\frac{\partial y}{\partial u} \tag{4.34}$$

2.4.7. Effect of the error feedback:

In equation (4.24) it is seen that the error feedback helps in improving the system response. We introduce an error feedback similar to the scheme by Kawato's and figure 4.8 shows the modified scheme.The input to the gain g is:

$$z = v + e = v + (y_d - y) \tag{4.35}$$

ity. In the case of MIMO systems, g becomes a matrix as in the case of K_k and K_c and need not be diagonal. The important differences between the other schemes and the gain layer scheme are: (i) The gain factor is adaptable and hence the initial g need not be estimated precisely; note that in the feedback gain scheme, the gain factor is fixed and need to be estimated precisely for a better tracking; (ii) The network weights are updated using the total system error rather than the transformed error as in the other schemes; this reduces the fluctuations in the weight space due to the on–line estimation of differential gain.

The network updating equations are derived as before and the the output layer updating rule becomes,

$$\delta_j^p = e_y \, f'_j(net_j^p) \tag{4.29}$$

The equations for other layers remain same as equation (4.13).

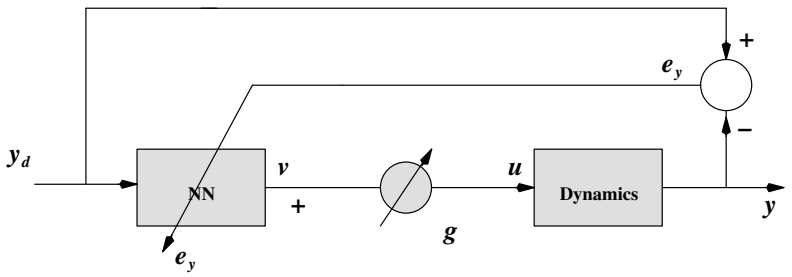

Figure 3.7 Gain Layer Scheme

2.4.6. Updating the Gain factor:

Let *g(k)* be the gain factor at the instant k. The adaptation rule for *g(k)* can be written as,

$$g(k) = g(k-1) + \Delta g(k) \tag{4.30}$$

network error directly affects the control input resulting in faster learning, while in other schemes, the error is reflected only through the network. It is worth pointing out that the network learning rate is usually kept very small when using the backpropagation algorithm for control applications, due to the stability considerations[51][101].

2.4.5. The Gain Layer Schemes:

Two different architectures for the direct control of dynamical systems using neural networks (Figures 4.7 and 4.8) are proposed here. In these schemes the total system error itself is used for updating the weights, and the effects of the differential gain of the dynamics is compensated using a gain layer. The control signal u is generated combining the system error feedback and the signal generated by the neural network. The network weights are updated using the backpropagation algorithm using the error e_y. The gain layer is initialized by a rough estimate of the inverse d.c. gain of the dynamics and is updated at each iteration using the gradient descent approach employed in the backpropagation. The objective function to be minimized is taken as as $E_y = \frac{1}{2} \| y_d - y \|^2$. The network weights can be updated using the total error e_y, with out any transformation, if $E_y \simeq E_v$ (see figure 4.8), there by avoiding the influence of the on–line differential gain estimation on the network updating. $E_y \simeq E_v$ when

$$\frac{\partial u}{\partial v} \simeq \frac{\partial u}{\partial y} \tag{4.27}$$

or,

$$g = J(u)^{-1} \tag{4.28}$$

The gain factor g is similar to the feedback gain constant K_k of Kawato's scheme and the inverse transfer matrix K_c of Chen *et al.*[98]. It can be initialized to a rough estimate of the steady state gain of the dynamics and can be adapted on–line for a better tracking and stabil-

With the approximation, $K_k \simeq \frac{1}{J(u)}$, substituted in equation (4.24.), we get

$$\frac{\partial E_y}{\partial w} = -K_k \frac{e_y}{2} \frac{\partial v}{\partial w} \tag{4.25}$$

and the resulting weight updating equation for the output layer is:

$$\delta_j^p = K_k \frac{e_y}{2} f'_j(net_j^p) \tag{4.26}$$

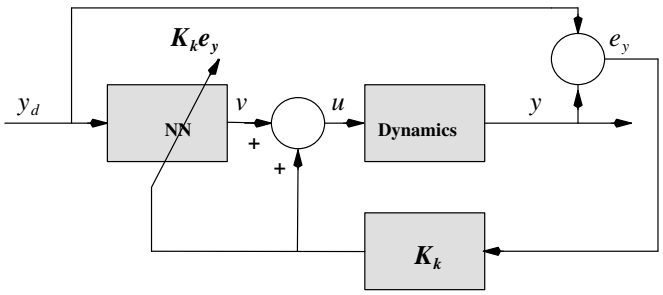

Figure 3.6 Feedback Gain Scheme

Comparing equations (4.26.) and (4.18), it can be seen that the feedback error scheme is equivalent to the inverse transfer matrix scheme when $J^{-1}(u) = \frac{K_k}{2}$. For SISO dynamics, with off–line estimation of the Jacobian, both methods provide similar results. However the schemes differ when the dynamics is MIMO. The inverse transfer matrix scheme is more general in the sense that it provides non–zero off–diagonal elements in the transfer matrix (providing the information about the channel coupling) while the feedback gain K_k is a diagonal matrix[98]. It may be noted that for good tracking performance over a wide range of nonlinear time–varying dynamics, sophisticated estimation of K_k is needed. Another important aspect of this scheme is the increased speed of learning due to the error feedback. The

$$u = K_k e_y + v \tag{4.19}$$

where $e_y = y_d - y$, as before.

If the instantaneous differential gain of the dynamics is defined as $J(u)$, the system response y can be expressed as:

$$y = [v + K_k(y_d - y)]\ J(u) \tag{4.20}$$

Expanding equation (4.20) and rewriting:

$$y = \frac{J(u)v}{1 + J(u)K_k} + \frac{K_k J(u) y_d}{1 + J(u)K_k} \tag{4.21}$$

With the approximation $K_k \simeq \frac{1}{J(u)}$, the equation (4.21) reduces to:

$$y = \frac{J(u)v}{2} + \frac{y_d}{2} \tag{4.22}$$

Note that the command signal y_d is directly influencing the system response because of the error feedback.

The error function to be minimized in this scheme is same as that of equation (4.17), and the network updating equations could be derived as before. From equation (4.20) it follows that:

$$\frac{\partial y}{\partial w} = J(u)\ [\frac{\partial v}{\partial w} - K_k \frac{\partial y}{\partial w}] \tag{4.23}$$

That is,

$$\frac{\partial y}{\partial w} = \frac{J(u)\ \partial v / \partial w}{1 + J(u)K_k} \tag{4.24}$$

(4.17), it can be seen that the differential gain of the dynamics affects the network updating in different ways in both the schemes. Since the dynamics is not taken as an additional layer, the convergence may be faster in the inverse transfer matrix scheme[98], provided K_c is computed off–line. An on–line computation of the inverse differential gain ($J^{-1}(u)$) produces fast varying fluctuations in the network weight space. Also, for control signals of constant values ($\partial u \simeq 0$), the on–line estimated values of the inverse differential gain may be inaccurate and unreliable.

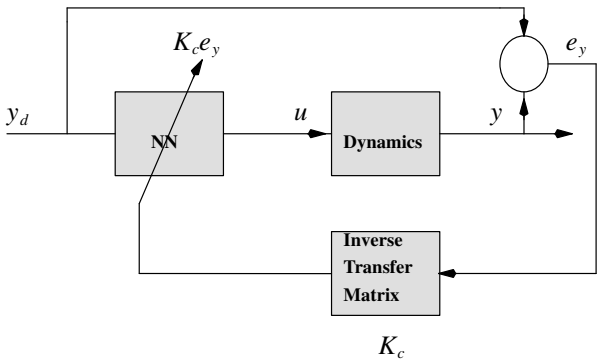

Figure 3.5 Inverse Transfer Matrix Scheme

2.4.4. The Feedback Gain Scheme:

Motivated by the voluntary movements in biological systems, Kawato *et al.*[99] suggest a feedback error scheme for the direct control of dynamical systems (figure 4.6). The feedback gain K_k is estimated off–line and is used for transforming the error for updating the network weights as well as for generating the control signal u. From figure 4.6, we obtain the following expressions for the control signal u :

$$E_{Ky} = \frac{1}{2} \| K_c e_y \|^2 \tag{4.14}$$

where K_c represents the inverse transfer relationship of the dynamics. The network updating is done by estimating the derivative of the error function with respect to the weights. That is,

$$\frac{\partial E_{Ky}}{\partial w} = - K_c^2 e_y \frac{\partial y}{\partial w} \tag{4.15}$$

As before $J(u) = \frac{\partial y}{\partial u}$ and equation (4.15) reduces to:

$$\frac{\partial E_{Ky}}{\partial w} = - K_c^2 e_y \; J(u) \frac{\partial u}{\partial w} \tag{4.16}$$

If we approximate $K_c \simeq \frac{1}{J(u)}$, E_{Ky} corresponds to the squared error at the NN output. Then

$$\frac{\partial E_{Ky}}{\partial w} = - J^{-1}(u) \; e_y \; \frac{\partial u}{\partial w} \tag{4.17}$$

The updating rules in this scheme remains the same as in Psaltis' scheme, except those corresponding to the output layer weights. For the output layer weights, equation (4.12) gets modified as;

$$\delta_j^p = J^{-1}(u) \; e_y \; f'_j(net_j^p) \tag{4.18}$$

The important difference between the method due to Psaltis *et al.* and the inverse transfer matrix scheme is that the first method minimizes the system error e_y while the second method minimizes error estimated at the output of the network, e_u. Comparing equations (4.10) and

$$\delta_j^p = J(u)\ e_y\ f'_j(net_j^p) \tag{4.12}$$

and for any other layer as

$$\delta_i^p = f'_i(net_i^p) \sum_j \delta_j^p w_{ji} \tag{4.13}$$

where f' denotes the derivative of the sigmoidal transformation function.

The differential gain *J(u)*(or instantaneous forward Jacobian) could be estimated on–line from the changes in y and u over two consecutive time instants. Alternately, the dynamics could be modeled using a neural network and could be used for transforming the error e_y to $e_{u,}$ the equivalent error at the neural network output (see figure 4.2) [26]. There is a specific modeling stage in this method. Troudet *et al*. discuss such a scheme for the flight control application in [100]. One of the problems with backpropagating the error through the dynamics is that as the output layer in the composite network architecture (the plant) is non–modifiable, the convergence may be slow[98]. From equation (4.12) it can be seen that the differential gain of the dynamics directly affects the network updating. An on–line numerical estimation using ratio of differences may generate large oscillations in *J(u),* which will be reflected in the network weight space. This problem is more dominant when the reference command signals are fast varying. Also, when the changes in inputs and outputs are very small (steady state conditions) numerically estimated Jacobian *J(u)*, will not be accurate. The inverse transfer matrix scheme suggested by Chen *et al*. [97][98] is a more relevant method under such conditions.

2.4.3. The Inverse Transfer Matrix Scheme:

In the inverse transfer matrix scheme (figure 4.5), the error function to be modified is defined as;

total error e_y with a transformation factor $\frac{\partial y}{\partial u}$. If we define $J(u) = \frac{\partial y}{\partial u}$, equation (4.9) could be rewritten as

$$\frac{\partial E_y}{\partial w} = - e_y \, J(u) \frac{\partial u}{\partial w} \tag{4.10}$$

The term $\frac{\partial u}{\partial w}$ corresponds to the output of the lower layer neurons[14].

2.4.2. Backpropagating the error through the dynamics:

If we consider the plant dynamics as an additional layer having interconnection strength *J(u)* with no updating done, equation (4.10) represents the backpropagation of the system error e_y through the plant. This approach originally proposed by Psaltis *et al*[52] and termed as 'specialized learning scheme', is an effective way for the direct control of dynamical systems using neural networks and is used in a number of later studies, in different forms[100][102]. The network updating equations are reviewed here for comparing with the other schemes. Following the notations in [14], let us define Δw_{ji}^p as the change in weight w_{ji} (in the p^{th} layer), connecting neurons *i* (in the lower layer) to the neuron *j* (in the higher layer); δ_j^p as the derivative of the error at the layer *p*, E_p, with respect to the summed input to the sigmoid transformation (net_j^p) at neuron j, and o_j^p as the output of the neuron. Then the updating equations for any layer could be shown to be[14],

$$\Delta w_{ij}^p = \eta \, \delta_j^p \, o_i^p \tag{4.11}$$

Using equation (4.10), we could get δ_j^p for the output layer of the network as

transformation to the neural network. The gain layer is initialized approximately to the steady state inverse gain of the dynamics and is adapted on–line. An error feedback, similar to the one in Kawato's scheme[99] is introduced to improve the dynamic response of the control system. Results of simulation studies performed on a nonlinear underwater vehicle dynamics, show that the presented scheme gives promising results. In the following sections the relevant scheme is described and the simulation results[62] are presented. Though Single Input Single Output (SISO) cases are used presently for the ease of explanation and comparison, the schemes can be readily extended to MIMO cases as well.

2.4.1. The network error and the system error:

Figure 4.2 shows the basic direct control scheme for the SISO dynamics. The NN learns the inverse characteristics of the dynamics implicitly to generate the correct control signal u, to drive the system state y to the desired state y_d. Defining the objective function to be minimized by the controller as the squared error at the output

$$E_y = \frac{1}{2} \| y_d - y \|^2 \tag{4.7}$$

If $(y_d - y)$ is defined as e_y, the gradient of the error with respect to the network interconnection strengths is

$$\frac{\partial E_y}{\partial w} = - e_y \frac{\partial y}{\partial w} \tag{4.8}$$

$$= - e_y \frac{\partial y}{\partial u} \frac{\partial u}{\partial w} \tag{4.9}$$

The term $\frac{\partial y}{\partial u}$ corresponds to the forward gain of the dynamics and is an important factor for the stability of the overall system[101]. The term $e_y \frac{\partial y}{\partial u}$ represents the transformation of the

2.4. THE BACKPROPAGATION ALGORITHM FOR LEARNING CONTROL APPLICATIONS

The direct control architecture shown in figure 4.2 does not impose any restriction on the learning algorithm used for updating the neural network weights. The objective of the scheme is simply to track the command signal to the desired level, while maintaining the overall stability of the system. One of the difficulties in implementing such a control scheme using the backpropagation is that the desired network output signal is not specified explicitly. Hence the error at the output of the network (network error) which is needed for updating the weights is not available (see figure 4.2). The system error *e* does not represent the error at the network output. At least three different ways are available to derive the network error from the system error. They are: (i) Considering the dynamics as an unmodifiable layer and backpropagating the error through the dynamics[26][52]; (ii) Transforming the total system error by multiplying it by the inverse transfer matrix[97][98]; and (iii) Implementing a fixed feedback gain which generates a feedback signal as well as a transformed error signal for network updating[26][99]. The differential gain of the dynamics (forward Jacobian) directly affects the network updating in methods (i) and (ii). An on–line numerical estimation of Jacobian results in sharp changes in the transformed error and hence in the network weights. On the other hand, method (iii) may give robust results only over a particular range, since the feedback gain is fixed.

In this section we review the above three implementations of direct control scheme using backpropagation algorithm and then propose a modified architecture[62]. In the proposed architecture, the network interconnection strengths are updated using the system error rather than the transformed error used in the other schemes. The numerical ill effects of the Jacobian of controlled dynamics are moderated by adding an extra layer (gain layer) with linear

the roll angle control were 0.003 and 1.0, respectively, while the weights were updated every 100 iterations. In both cases, the network had an input layer of 2 neurons, two hidden layers each of 10 neurons and an output layer of one neuron, and the interconnection weights were initially set to small random values between –0.01 and +0.01. Parameters such as the number of layers and neurons in each layer, the step size and the batch size were determined empirically to get a close tracking of the command signal.

The network was found to perform better when the command signals were slow varying, which is the case in actual situations. The stepsize along with the learning rate determine the time constant of the network and are critical factors in the stable operation of the overall system. The stability is also related to the initial distribution of weights. It is necessary to set the initial weights to very small random values to ensure stability. Because of the stochastic nature of the Alopex algorithm, a perfect tracking with zero error may not be possible but a close enough tracking can be obtained by choosing proper network parameters.

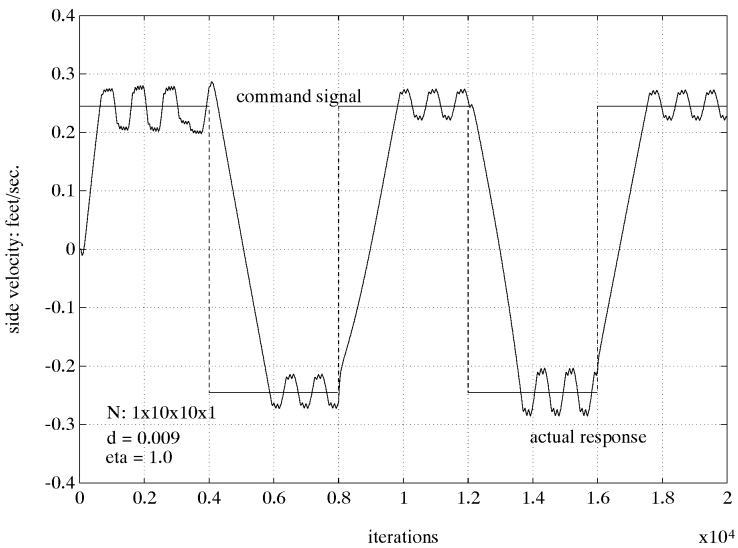

Figure 3.4 Tracking of the vertical velocity command signal

2.3.2. Results:

In the simulation studies, the squared error at the output is taken as the function to be optimized and the weights of the neural network are updated using the Alopex algorithm[56]. As shown in figures 4.3 and 4.4, rectangular waveforms are applied as command signals so as to incorporate all the frequency components. The actual responses of the dynamics, for the control of side velocity and roll angle, are also shown along with the command signals. It can be seen that the control signal provided by the neural network, drives the actual response close enough to the desired one, in both cases.

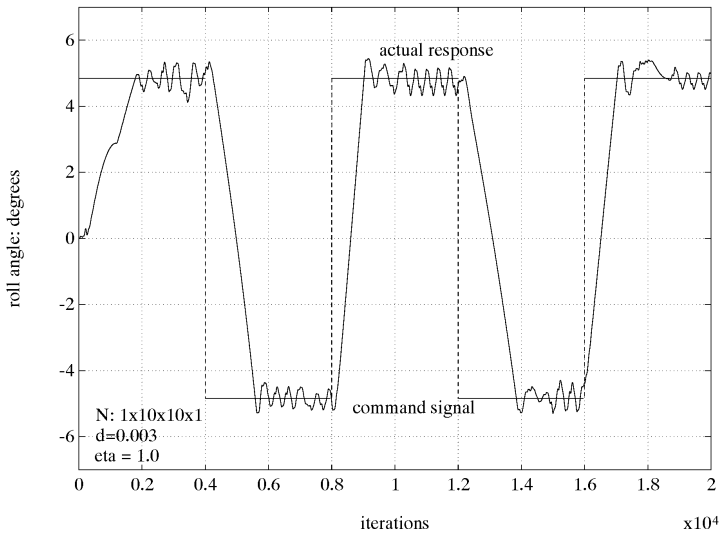

Figure 3.3 Tracking of the roll angle command signal

In the case of side velocity control, the weights were updated every 50 iterations(batch size) with a stepsize of 0.009 and learning rate of 1.0. The stepsize and learning rate used for

As mentioned in the previous chapter, one of the distinct advantages of the Alopex algorithm is that an explicit error derivative is not required for updating the network weights. The error over a period of time is accumulated and the weights are updated according to this error (batch updating).

2.3.1. The Vehicle Dynamics

The linearized dynamics of a six degree of freedom (DOF) underwater deep submersible rescue vehicle (DSRV) was used for the studies. The vehicle dynamics has two input parameters, rudder and stern plane deflections which control six output parameters, viz., three linear velocities and the three angular velocities. The rudder deflection affects two angular velocities, roll (ϕ) and yaw(ψ), and a linear velocity, that along the y–direction (v). The coordinate descriptions and the nonlinear equations of motion of the DSRV are given in Appendix C. The linearized version of the input–output relationships at a specific operating condition for the two cases studied are given below in the form of their system functions[95][96]. The forward velocity of the vehicle is assumed to be the designed maximum of 10.13 feet/sec.

$$\frac{v}{\delta_r} = \frac{(0.5829s^3 + 0.769s^2 + 0.351s + 0.3384)}{(0.8459s^4 + 1.0033s^3 + 0.6272s^2 + 0.4352s + 0.0637} \quad (4.5)$$

$$\frac{\phi}{\delta_r} = \frac{(0.1461s^2 + 0.2340s + 0.0910)}{(0.8459s^4 + 1.0033s^3 + 0.6272s^2 + 0.4352s + 0.0637} \quad (4.6)$$

where v is the side velocity, ϕ the roll angle and δ_r the rudder deflection. v and ϕ are in feet/sec. and radians, respectively, when δ_r is expressed in radians.

2.3. ALOPEX ALGORITHM FOR THE LEARNING CONTROL OF DYNAMICAL SYSTEMS

Present studies are conducted to investigate the performance of feedforward networks, trained using the Alopex algorithm, for on–line learning control applications. The control scheme used is shown in figure 4.2 The neural network has two input nodes, one for the desired response for the vehicle and the other one for the delayed system response. More than one delayed output and the command inputs could be used as the input to the network, which may improve the tracking performance of the system. In the current studies, no noticeable improvement was observed when the number of inputs are more than two. The network produces the control input u which drives the states of the controlled system to the desired ones. The network weights are updated using the squared error at the output of the dynamics. Note that in a backpropagation algorithm based method, the network weights cannot be updated using the system error *'e'* as it does not represent the error at the output of the network. This limitation and its effects on the on–line learning control applications are discussed in the following section. Also, only the control of linear dynamics is discussed in this section. Simulation results on a nonlinear dynamics[62] are discussed in Chapter 7.

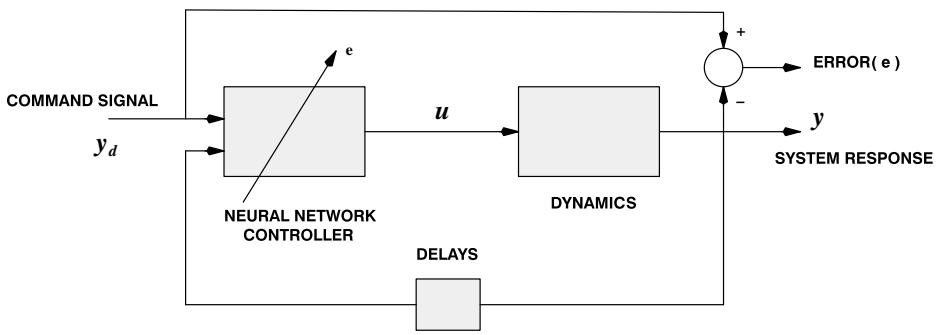

Figure 3.2 The control scheme

linear system with the ability to adapt itself according to a performance index, which in turn related to the training algorithm used. Hence with a suitable training scheme, the encoding and classification stages in figure 4.1 could be mapped into a neural network. For a sigmoidal neuronal transformation function, the classifier output would be continuous in the range [–1, 1.0].

Most of the neuro–control strategies are implemented in the context of adaptive control. Even though both adaptive control and learning control terminologies are used in a common context in neural network literature, there are subtle differences between both the schemes. A controller emphasizing continuous monitoring of its own performance could be termed as an adaptive controller rather than a learning controller. All adaptive controllers are learning controllers but not vice versa[78][94].

2.2.1. On–line and Off–line Controllers:

The control schemes described in the earlier sections are extensions of the static pattern classifying schemes. The training and classification (control) stages are distinct in these schemes. The training set has to contain samples from all the possible control situations for an effective post–training operation of the controller. This is called an off–line controller. When the learning and control actions are on–going processes or on–line, the controller architecture has to account for the temporal variations in the input signals which can be seen as the pattern recognition problem of dynamic signals. The efforts described in this dissertation deal only with the on–line learning control schemes.

Then the control signal $u = \begin{cases} +1 \text{ if } f(V) < 0 \\ -1 \text{ if } f(V) > 0 \end{cases}$ (4.3)

where $f(V) = V^T W$ (4.4)

It can be shown that[78] such a controller architecture will approximate a switching surface of the form $f(x_1, x_2, ..., x_k) = 0$. The training set consists the population of the control situations normally expected and is used for estimating the weight vector *W*. The trained controller can be used to classify different control situations.

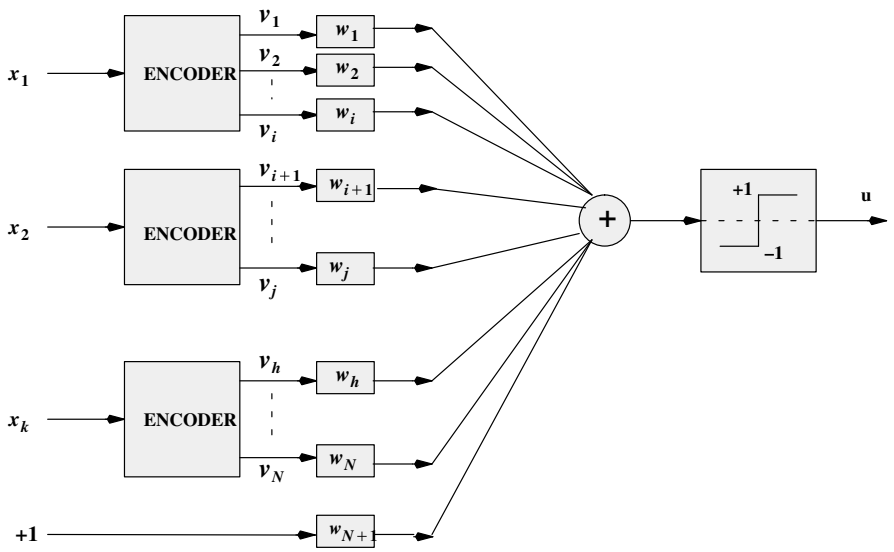

Figure 3.1 Learning controller using linear discriminant function (adopted from [78])

2.2. THE NEURAL NETWORK BASED LEARNING CONTROLLERS

From the control system point of view, feedforward neural networks represent static nonlinear mapping between the inputs and the outputs. The network could be considered as a non-

Thus learning control task involves system identification, where the information about the controlled system is gathered over a period of time. This may be done either through an explicit system identification stage, or implicitly through procedures where the identification is a part of the control action. Respective examples are indirect and direct adaptive control schemes[85].

Let us consider a pattern recognition problem in which m possible pattern classes $c_1, c_2, c_3, ..., c_m$ are to be classified from a k dimensional feature vector input $x_1, x_2, x_3, ..., x_k$. The classification operation, generally referred to as mapping the input space Ω_X into the decision space $\Omega_{c,}$ involves the partitioning of the k dimensional input space into m mutually exclusive regions. A classification $x \in c_i$ is made if the discriminant measure $d_i(x) > d_j(x), \ \forall\ j = 1, 2, ..., m; \ j \neq i$. Note that the discriminant measure could assume any form, such as linear, polynomial or probabilistic.

The equivalence of such a pattern classifier to a learning controller could be investigated in the following way: If the partitioning of feature space Ω_x regarded as equivalent to the partition of state space, the decision surface in feature space is equivalent to the switching surface in the state space[26][78]. Hence the learning controller is similar to a pattern classifier. The regions in the state space correspond to various control situations. The schematics of such a learning controller is shown in figure 4.0 The state inputs are represented by $x_1, x_2, x_3, ..., x_k$ and $w_1, w_2, ..., w_N$ denotes the parameters to be learned. The k dimensional state vector X is first transformed into elementary control situations which form the N dimensional pattern vector $v_1, v_2, ..., v_N$. Let

$$V = [v_1, v_2, ..., v_N, 1]^T \tag{4.1}$$

and

$$W = [w_1, w_2, ..., w_N, 1]^T \tag{4.2}$$

One approach to incorporate the information about time into the learning, is to represent it as an explicit dimension in the input space. The sonar target recognition problem discussed earlier is an example of this approach. The second approach is to allow the time to be represented by the effect it has on information processing[92][93], or in other words, build memory into the network. An example of this approach is the recurrent architecture. A number of similar approaches and examples, mainly in the context of speech recognition, could be found in the literature[47][48][72][93]. Pertinent details are discussed in Chapter 6. The objective of the present chapter is to extend the previously discussed temporal pattern classification studies to learning control applications. The essential similarities and differences between pattern classification and learning control processes are reviewed. The application of the Alopex algorithm on learning control tasks is discussed in the initial part. The approaches using the backpropagation algorithm are reviewed and two new modified schemes for the learning control schemes using the backpropagation are proposed.

2.1. PATTERN CLASSIFIERS AND LEARNING CONTROLLERS

Often, learning can be deduced to the process of estimating or successively approximating a set of model parameters describing the process to be learned[78]. When the objective of a learning system is to seek the model which matches closely a physical system, it is referred to as the system identification. Depending on the type of information usage, the procedure may identify the forward relationship or the inverse relationship of the physical system. A necessary requirement for any estimation process is the availability of enough training samples which is capable of characterizing the input–output behavior of the model over a variety of inputs.

CHAPTER 2.

LEARNING CONTROL OF DYNAMICAL SYSTEMS USING FEEDFORWARD NEURAL NETWORKS: PART I

In the previous chapter, the effectiveness of the Alopex as a learning algorithm was demonstrated on a number of static and temporal pattern recognition tasks. Even though the static recognition tasks are adequate to a certain extent for performance comparison of different learning algorithms, they cannot fully characterize the learning process with temporal recognition tasks. For example, consider the case of the learning static and temporal XOR patterns. Even though the objective of the neural network is to learn the logical operation embedded in the input patterns (which are same in both the cases), the relevant problems pose entirely different complexities. Time is an important parameter in the second case. In this case, the network has to keep information about the previous states of the neurons or about the history of the input signal.