error function on a number of problems. Figure 3.19a–d shows the error surface around the solution point for the XOR network, using the Alopex algorithm[50][55]. It could be seen that the surface for the log error is much smoother than that for the squared error. In an earlier study[50][55], it has also been found that, on average, networks trained with information theoretic error function gets 'stuck' at a local minima only 4% of the time while a network with squared error function gets 'stuck' in 19% of the trials. These results show that for specific classes of problems, smoother error surfaces, with fewer local minima, could be obtained by choosing an appropriate error measure.

We also demonstrated the ability of the Alopex algorithm on network architectures having asymmetrical feedback connections. The algorithm was able to classify temporal XOR sequences of delays 0, 1, and 2, using such a network architecture.

MONKS problems. On all the three problems, the Alopex could produce 100% correct classification. These are hence the best reported results so far on the MONKS problems. The learning ability of the Alopex algorithm is also studied on two practical problems, viz., the handwritten character recognition and side–scan sonar target detection.

The usefulness of choosing a proper error function was studied and the results show that an information theoretic error function gives better results than the traditionally used squared

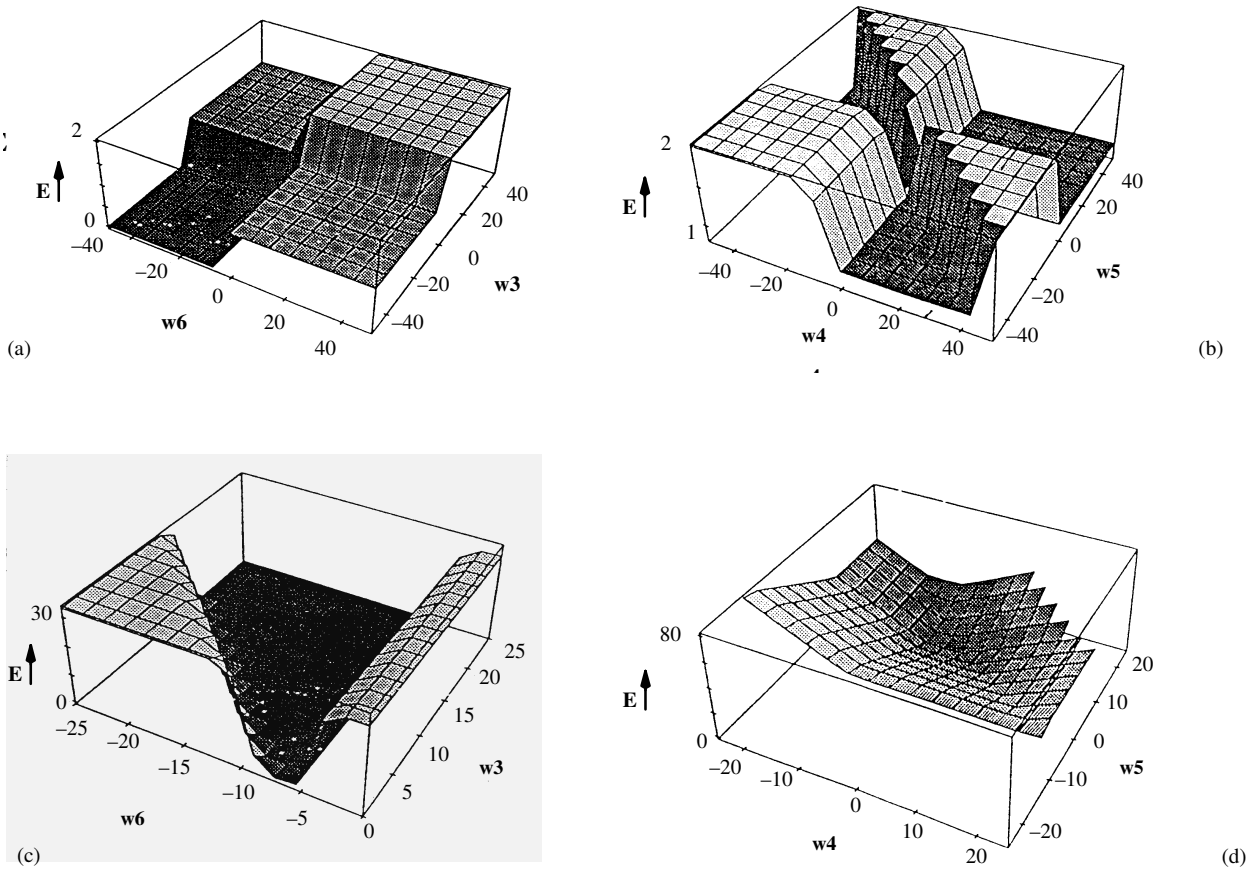


Figure 3.20 Error surfaces for 2−2−1 XOR problem (from [55]). The network architecture is shown in figure 3.1.

(a) Square error surface with respect to the weights w3 & w6

(b) Square error surface with respect to the weights w4 & w5

(c) Log error surface with respect to the weights w3 & w6

(d) Log error surface with respect to the weights w4 & w5

the output of a 1x4x1 network after 4,668 iterations. Again, this network has learned almost perfectly the XOR problem with delay 1. Figure 3.19. shows the output of a 1x6x1 network (after 27,000 iterations) plotted along with the segment of the test data. The average error per pattern as a function of iterations is shown in figure 3.20.

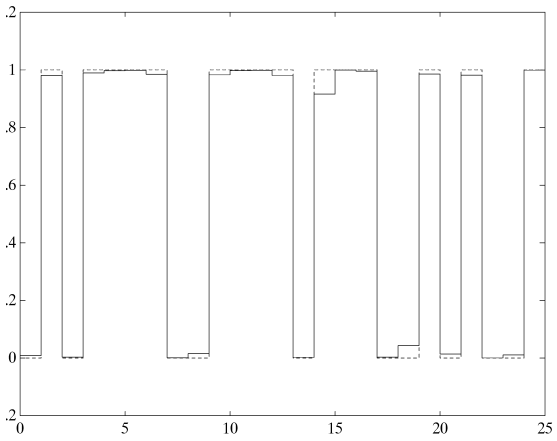


Figure 3.18 Output of a 1x6x1 network trained to solve τ=2 temporal XOR problem.

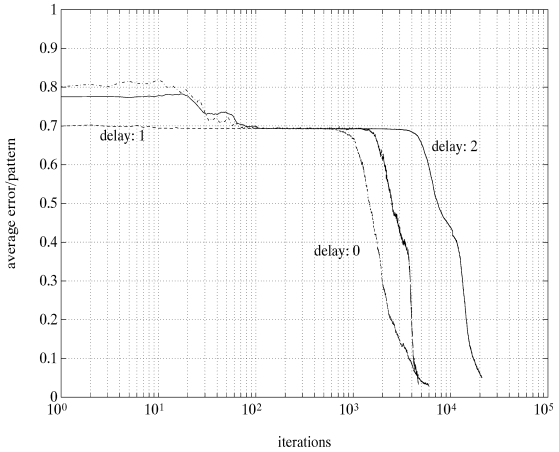


Figure 3.19 The average error per bit for the training set for the three temporal XOR problems.

3.7. SUMMARY

The effectiveness of the Alopex algorithm as a learning algorithm for neural networks is demonstrated on a number of benchmark as well as practical problems. The results on Parity, Symmetry and Encoder problems show that the Alopex requires comparable number of iterations as backpropagation. It should be noted that, since in the Alopex the updating is synchronous, actual computation time required would be much less than in the case of asynchronous algorithms such as the backpropagation. The generalization ability of the Alopex algorithm, even when the patterns are misclassified, is demonstrated in respect of the

The network contains one input unit, one output unit and various number of hidden units. The task is to produce at the output unit, the XOR of the current input and the input from previous time–step. So the network needs to internally store input information for one time–step to solve the problem. The task can be made more difficult by insisting that the current output should be the XOR of inputs from time t–τ and t–(τ+1). Here, the network needs to store vales from many time–steps in the past. The networks are taught to learn the temporal XOR problem with τ = 0, 1, and 2. Figure 3.16 shows the schematic of a relevant network with an input unit, an output unit and four hidden units. A randomly generated binary string 3000 bits long was used for training and another one 100 bits long was used for testing.

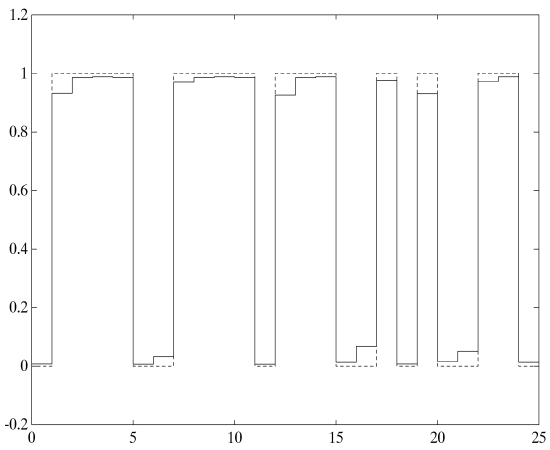


Figure 3.16 The output of a 1x2x1 network trained to solve τ=0 temporal XOR problem.

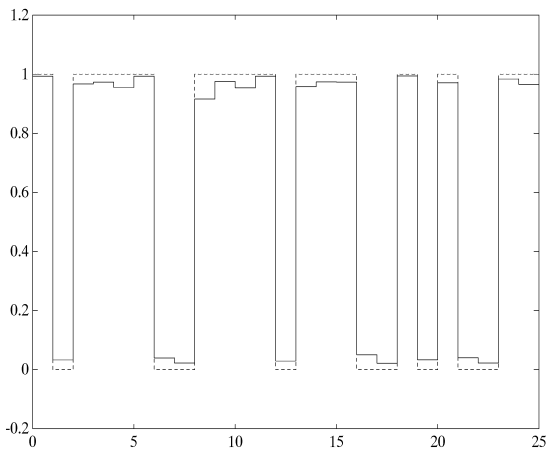


Figure 3.17 The output of a 1x4x1 network trained to solve τ=1 temporal XOR problem.

Dashed line represents the test sequence and solid line represent the output of the network.

A network with two hidden units was used to learn the problem with no delay (τ =0). Figure 3.17 shows a segment of the test string and the output of the network after 6,000 iterations. The network has learned successfully the problem. Figure 3.18 shows the test segment and

3.6.2. Recurrent Network architecture: Classification of Temporal XOR:

The learning capabilities of the feedforward networks with sigmoidal transformation functions have been comprehensively studied[83][84]. But in applications involving temporal signals and dynamical system, capabilities of such networks are rather limited. Also, the application of neural networks on dynamical problems is valid exactly only when the networks themselves are dynamical in nature[51]. In this section, we demonstrate ability of the Alopex algorithm for training recurrent networks. We have used a network where the hidden layer neurons are totally interconnected (including self loops) to solve the temporal XOR problem with different delays.

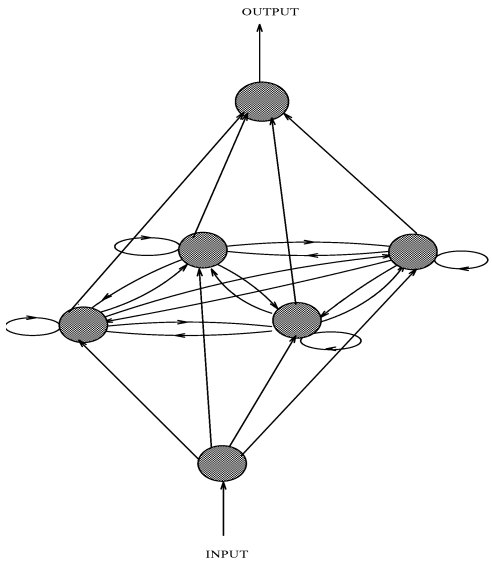


Figure 3.15 Recurrent network architecture to solve temporal XOR Problem

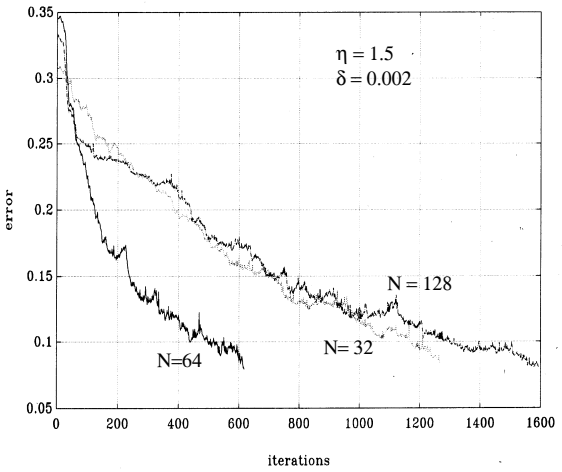


Figure 3.12 Change in the error convergence when the number of neurons in first hidden layer is changed.

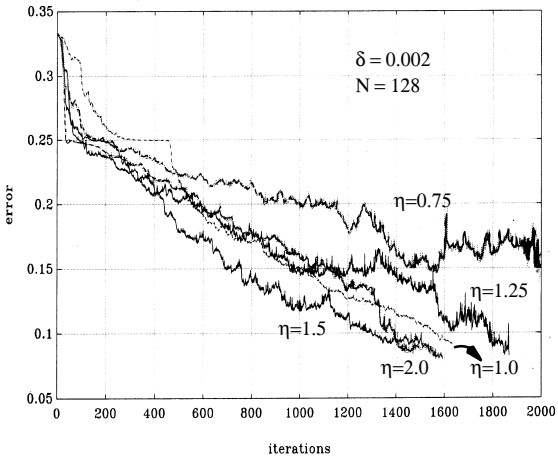


Figure 3.13 Change in the error convergence as the learning rate η is changed.

Best performance was obtained with 2 hidden layers; 64 neurons in first hidden layer and 2 in the second. No performance improvement was observed when the number of hidden layers was more than two. The best convergence was obtained for a stepsize of 0.002 and learning coefficient 1.5, with a maximum allowed error of 0.08 per output neuron. The network convergence was independent of the initial weight distribution. The network reached the global minimum even when all the weights were set to zero initially.

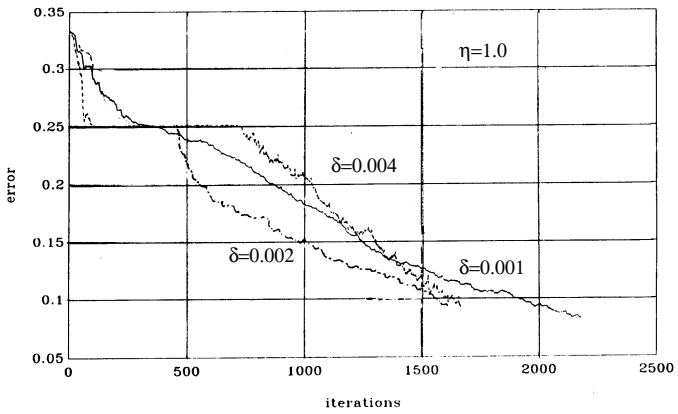


Figure 3.14 Change in the error convergence as the stepsize is changed

corresponding lower layer groups. The decisions of the second hidden layer neurons are combined in the output layer which consists of two neurons corresponding to the two classes of targets. In effect, the neural network scans a horizontal section of the sea bottom corresponding to five time frames. The present architecture was selected assuming that sufficient information about the object is contained in five consecutive returns.

The simulation data consists of the side–scan sonar returns from two targets, a metal drum and a wooden box, from a height of about 100 meters. Two sets of simulation data, one for each target, were created from the envelopes of two actual returns. The envelopes were first modulated with sine wave and then random noise was added to create the training set consisting of 32 returns, 16 in each set. A separate set of 8 returns was generated for the classification testing (returns from only one aspect angle were considered in this study). Each return is Hanning windowed, with an overlap of 16 points, and a 64 point FFT is calculated. The squared absolute values of the FFT are used as an estimate of the Power Spectral Density (PSD) envelope for each return. The PSD values are normalized to be between 0.0 and 1.0 and each sample in PSD envelope forms the input to a neuron in the input layer of the network.

The algorithm was tested for the convergence rate and classification accuracy for different number of hidden layers, number of neurons in hidden layers and initial weight distribution. The error variation with respect to the number of neurons in the first hidden layer is shown in figure 3.13. The learning rate and the stepsize were 1.5 and 0.002, respectively, for this case. Figure 3.14 shows the effect of varying the learning rate η on the error, for a fixed stepsize δ of 0.002. The error convergence as a function of the step size δ for a constant value of η, is shown in figure 3.15. The architecture has 64 neurons in each group of the first hidden layer, and two neurons in each group in the second hidden layer.

same material can have non–distinguishable returns, if the profiles of the targets seen by the sonar at that instant of scanning are similar. Hence, the network should be able to retain information about the previous returns for a correct classification. A network architecture with five delayed input sets (figure 3.12) was used for the present problem. See [47] for a similar network architecture used in the context of speech recognition.

The input layer consists of five groups corresponding to five consecutive sonar returns in the time frame. Each group in the input layer consists of 32 neurons, corresponding to the 32 samples of the PSD envelope. The first hidden layer consists of 4 groups of 64 neurons each having input connections only from two adjacent groups in the input layer. This enables the network to learn the temporal information contained in two consecutive returns. The second hidden layer consists of 4 groups of 2 neurons each and has interconnections only from the

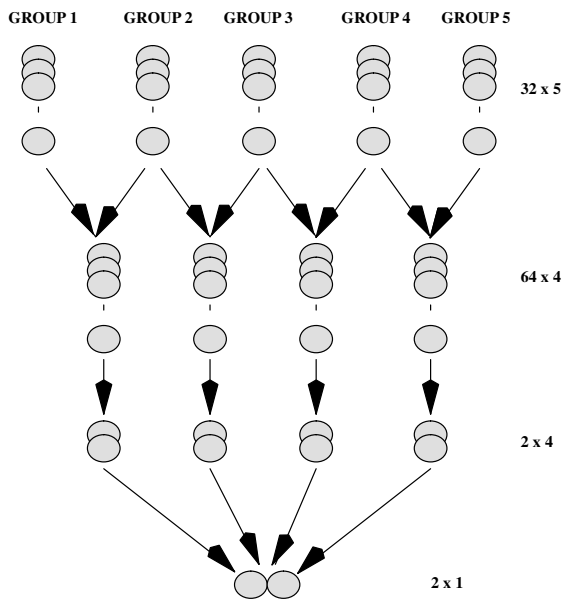
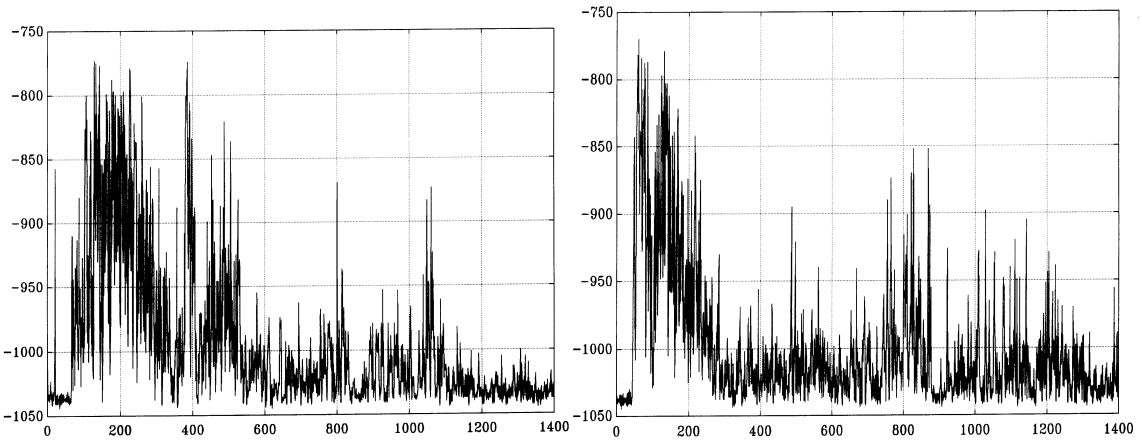
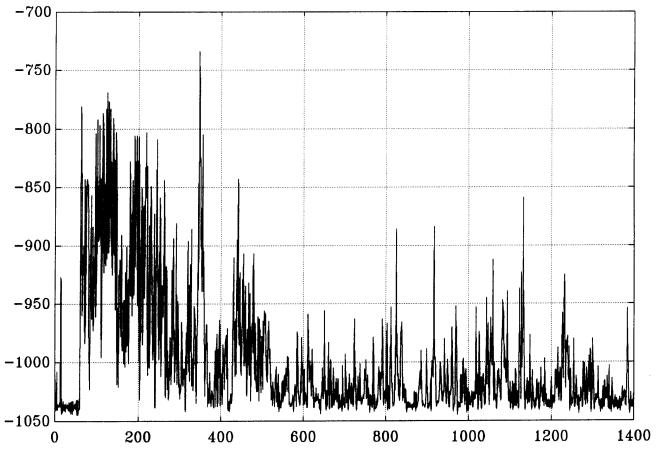


Figure 3.11 The neural network architecture for sonar target recognition

human involvement is currently necessary because of the complex and noisy nature of the sonar returns[91]. The sonar returns from two objects, a wooden box and a metallic drum along with the background signal are shown in figure 3.11a–c [59][60]. The complexity of the problem is evident from the similarity of the returns. Note that two different targets of



(a) (b)



(c)

Figure 3.10 Side−scan sonar returns
(a) from the wooden box
(b) from the metallic drum
(c) from the background

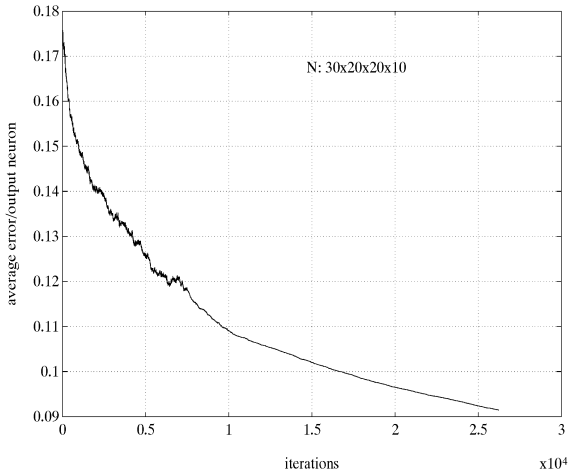


Figure 3.9 Convergence of the output error during training

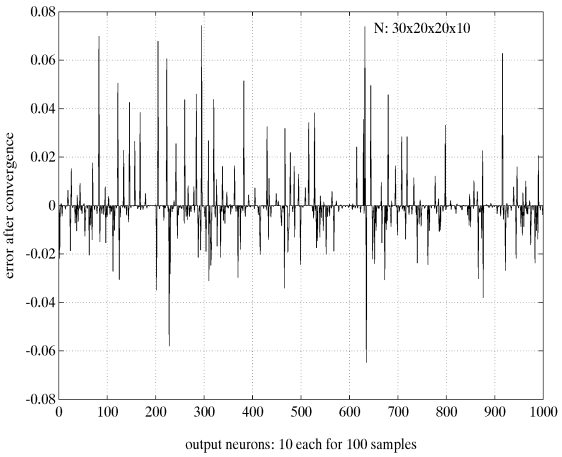


Figure 3.9 Output error after convergence over all the samples

3.6. LEARNING AND CLASSIFICATION OF DYNAMIC PATTERNS

In the studies described in the previous section, the input patterns are uncorrelated with each other and no temporal information is present in the input data. Learning becomes more complex when the network has to learn information about the previous input states, for correct classification. Two common neural network architectures are chosen presently for studying the effectiveness of Alopex on dynamic pattern recognition, namely 'parallelizing the time' approach and the recurrent network architecture.

3.6.1. 'Parallelizing the time' Approach: Recognition of underwater objects from side scan sonar returns:

The capability of networks trained with the Alopex algorithm are demonstrated using an undersea target recognition problem. It may be noted that in this practical problem extensive

The Fourier descriptors form the input to the neural network classifier. The network was shown all the 30 descriptors for each digit. 100 samples from the training set were presented sequentially and the training continued till the output error became less than 0.1.

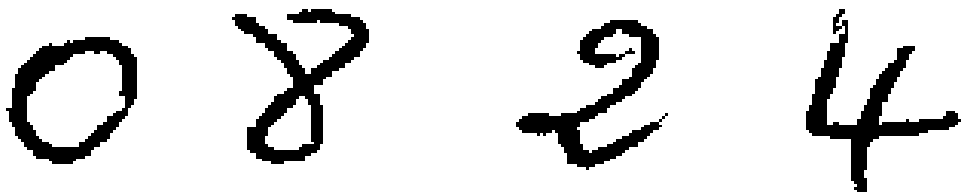
The training of the network was done with 100 samples 10 each from the 10 different classes, selected at random. The network was presented with the FDs for the 100 samples and was allowed to converge in all the patterns in all the samples. The step–size for adaptation of the weight and the number of layers and number of neurons in each layer are adjusted to get an optimum error convergence. The best convergence was obtained with a step size 0.005 and learning rate of 1.0. Figure 3.9 shows the convergence of the network in this case. The error corresponds to the average error per neuron per sample at the output. The final converged patterns in all the 100 samples are compared with the target patterns and figure 3.10 shows the error at each output neuron over the entire length of 100 samples. It can be seen that the convergence rate is 100% with an error less than 0.08. The converged network was tested with 50 digits, 5 from each class, which do not belong to the training set. The network was able to classify 49 of the presented digits correctly.

Table 3.3 Fourier Descriptors for the digits shown in figure 3.8.

digit	'0'		'8'		'2'		'4'	
FDs	A_n	λ_n	A_n	λ_n	A_n	λ_n	A_n	λ_n
1	0.397391	4.913768	1.043340	0.003838	0.899992	5.812483	0.156129	0.397739
2	4.473992	5.788829	0.583478	5.694290	2.123099	4.365014	1.042735	4.473992
3	4.974376	1.023792	1.362550	5.025680	0.505276	4.073431	0.063147	4.974376
4	0.341206	4.084522	0.243549	2.471761	0.664876	1.283149	1.121947	0.341206
5	4.732828	0.747564	0.129215	5.860723	0.430750	2.166913	0.233035	4.732828
6	3.665055	5.348615	0.326039	3.514683	0.259116	5.794081	0.166677	3.665055
7	3.761235	1.056302	0.304407	3.523595	0.275651	5.489411	0.048544	3.761235
8	6.049779	6.210600	0.108860	3.842446	0.241269	1.884809	0.289675	6.049779
9	2.356178	5.891035	0.188501	3.036107	0.216520	4.412789	0.157144	2.356178
10	4.452450	4.914101	0.223674	2.276665	0.295206	1.914344	0.125147	4.452450
11	4.962071	2.515147	0.123568	6.201783	0.041615	5.619387	0.067688	4.962071
12	4.712157	2.711829	0.143159	4.238128	0.142797	0.485972	0.166668	4.712157
13	5.113578	2.595903	0.058676	0.527760	0.202963	5.889127	0.115988	5.113578
14	0.944651	3.774034	0.058071	2.487405	0.161343	3.873921	0.020803	0.944651
15	2.879648	0.861592	0.052779	1.620927	0.141194	4.512780	0.175954	2.879648

The NIST database of hand printed digits was used for the simulation studies. The training set consists of 100 digits, 10 from each class. The training set is selected such that it contains diverse samples such as 'broken' '8' (without the upper loop as in figure 3.8b), 'broken' '9's and '2' with and with out loops. The orientation of the digits varies considerably so that only a rotation invariant recognition method could classify them correctly[63]. The Fourier Descriptors (FD) of the digits are estimated by following the boundary points keeping the inside region of the image to the right and using Zahn and Roskies equations[89] for polygonal curves. Only the first 15 descriptors are used for the recognition. Figures 3.8a–d show samples of four digits and their FDs are tabulated in table 3.3. It should be noted that no preprocessing such as smoothing or thinning is done on the digits and the FDs are calculated directly from the raw digits, thus a considerable speed enhancement is achieved.

The network has four layers, having 30 input neurons (15 each for the amplitude and phase descriptors), two hidden layers of 20 neurons each and the output layer having 10 neurons. The 10 output neurons correspond to the 10 classes to be recognized. It was experimentally found that the input set–size of 30 gives the best classification accuracy without increasing the number of network interconnections. The selection of hidden layers and the number of neurons in the layer was determined empirically by observing the convergence time as well as the accuracy of classification.



(a) (b) (c) (d)

Figure 3.8 Examples from the training set. Corresponding FDs are shown in Table 3.3.

It should be stressed here that 100% correct classifications were obtained using the Alopex algorithm for all the three MONKS problems. This is the best reported results on MONKS problems. The backpropagation algorithm could classify only 97% of the third test set correctly[88], while many of the machine learning algorithms could get 100% correct classifications only in one out of the three problems.

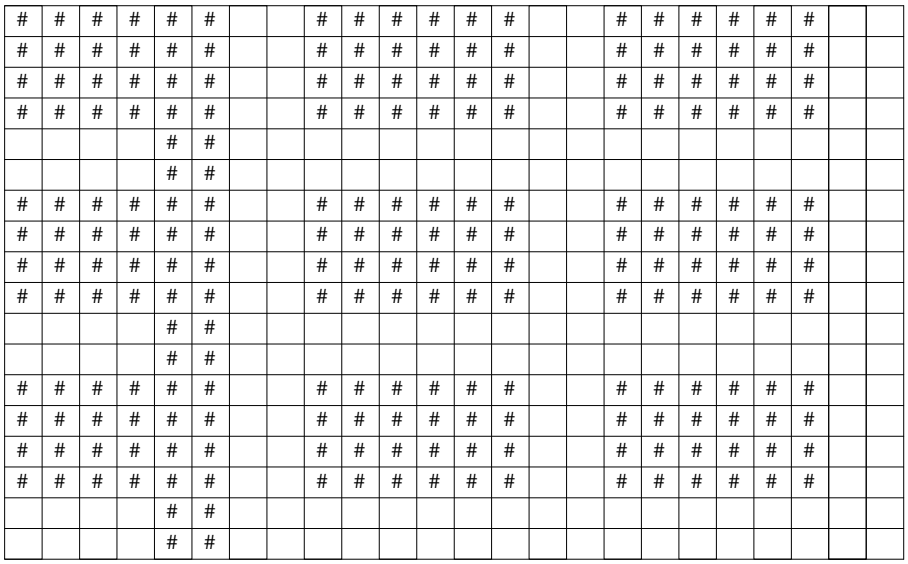


Figure 3.7c. Output of the network for the test pattern 3

3.5.5. 2D object recognition using Alopex algorithm:

In this section, we describe application of the Alopex algorithm as a classifier to recognize handwritten digits. The Fourier Descriptors of the object boundary[89][90] are taken as the features which form the input to the neural network. A multilayered perceptron architecture similar to those used in the earlier studies is used for these experiments[57][63]. A detailed description about the estimation of Fourier Descriptors and different neural network based object recognition schemes is given in Appendix A.

These problems were used for extensive bench–marking of different learning techniques, including those from the machine–learning field and neural networks (see [88] for details). Some of the machine learning techniques classified the first and third sets with 100% accuracy, but not the second. Three layer feed–forward networks trained using backpropagation (with weight decay) classified the first two sets with 100% accuracy, but not the third.

The Alopex algorithm was tried on MONK's problems using a feed–forward network with 15 input units, 3 hidden units and an output unit and the squared error as the objective function[50]. The test data set of the first problem was classified with 100% accuracy after 5000 iterations. figure 3.7a. shows the responses of the network for the test samples. The second test data set was correctly classified after 10,000 iterations. Figure 3.7b shows the responses of this network. The third test data set was classified correctly after 1000 iterations through the training data and the results are shown in figure 3.7c. The same problem can be solved using the log error, after only 662 iterations. This again shows the effectiveness of using the log error measure for faster convergence.

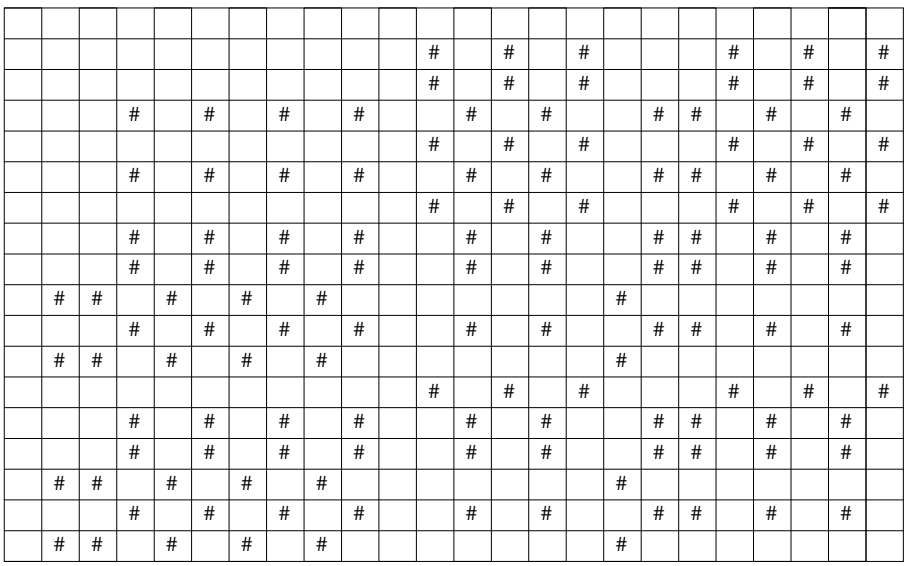


Figure 3.7b. Output of the network for the test pattern 2

Among the six discrete–valued attributes $x_1 - x_6$ that describe a problem, one attribute (x_5) has four possible values, three attributes (x_1 , x_2 ,and x_4) have three possible values and two attributes (x_3 , and x_6) have two possible values. For the first problem, the correct class is given by the logical description $x_1 = x_2$ or $x_5 =$ first value. For the second problem, the correct class is given by the description 'two of the six attributes have their first value' The correct class for the third problem is given by the description '($x_5 =$ third value and $x_4 =$ first value) or ($x_5 =$ fourth value and $x_2 =$ third value)'. Problem 1 is in standard disjunctive normal form. Problem 2 is similar to parity problems and combines different attributes in a way which makes it complicated to describe in disjunctive or conjunctive normal forms using the given attributes only. Problem 3 is again in disjunctive normal form. Of the 432 possible examples, 124 randomly chosen ones were designated as training samples for the first problem, 169 for the second problem and 122 for the third problem. The training set for the third problem also contained about 5% misclassifications. The entire set of 432 samples were used for testing.

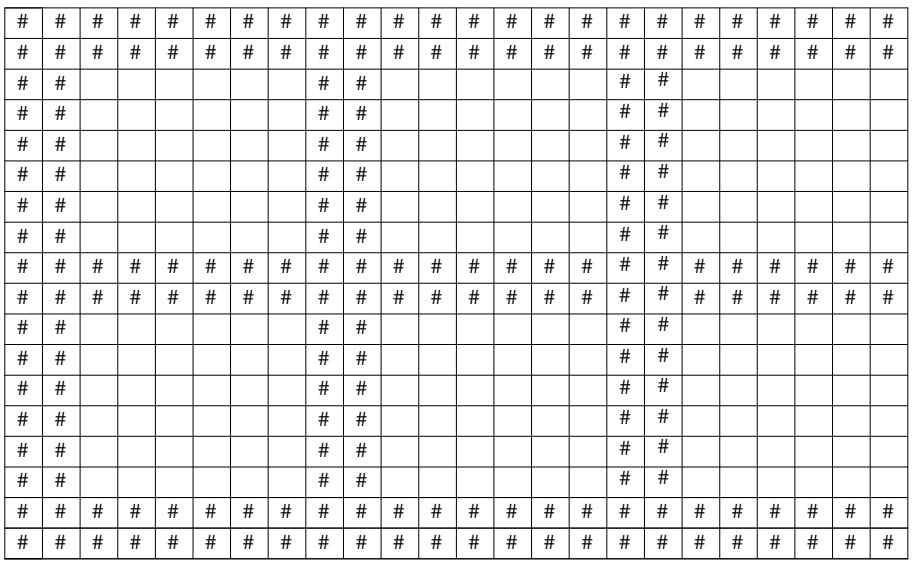


Figure 3.7a. Output of the network for the test pattern 1. See [88] for details on the grids and notations.

Table 3.2: Encoder problem: average number of iterations taken for convergence by Alopex algorithm

SIZE OF INPUT	SQUARE ERROR	SQRT. ERROR	LOG. ERROR
4 bits	1, 734	1, 229	1, 775
8 bits	6, 771	18, 662	13, 077
16 bits	no convergence	no convergence	20, 280
32 bits	no convergence	no convergence	75, 586

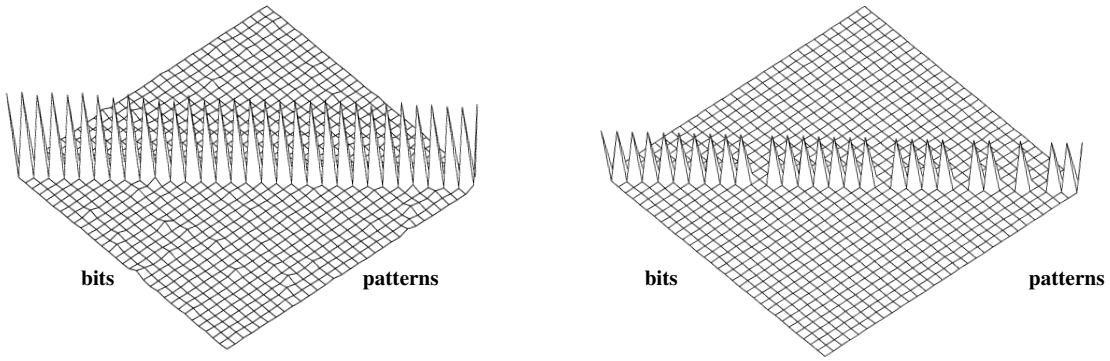


Figure 3.6 Recognized patterns after the convergence for encoder problem

(a) using log error.
(b) using square error

3.5.4. MONKS Problems:

These problems were initiated by Thrun *et al.* in 1991[88] to evaluate various learning techniques. In these problems, each training example is represented by six discrete–valued attributes. Each problem involves learning a binary function defined over this domain, from a set of training examples of this function.

momentum rate of 0.9 were used for BP. The performance of backpropagation was superior to that of Alopex in this particular problem [table 3.1]. The Alopex being a stochastic algorithm, may not perform as well as 'steepest descent' algorithms, for applications where a prominent global minimum is present and local minima are absent. It may be pointed out that, in Alopex all the weights are updated synchronously and hence with a parallel implementation scheme, the performance of the algorithm would be much better than that for BP, as the computational time taken per iteration would be much less. Figure 3.5 shows the error at the output layer as the network converges, in the case of Alopex algorithm. Again the log error function gives a better convergence characteristics than square root or square error functions.

3.5.3. Encoder Problem:

The scaling properties of the Alopex algorithm were investigated using the encoder problem. The input patterns were mapped on to the hidden layer and mapped back to the original pattern as the output. The number of neurons in the hidden layer is $log_2(N)$ where N is the number of bits in the input patterns. The average number of iterations (over 25 trials) taken for 4, 8, 16 and 32 bit encoders, using the three different error measures are shown in table 3.2. For 16 and 32 bit encoders the network did not converge with square and square root errors (the error was observed till 200,000 iterations). But a network using the log error converged in all cases. These results show that with appropriate error measures, Alopex algorithm can be used for learning in problems having large number of output classes. The final converged patterns for the 32 bit encoder are shown in figure 3.6. The network with squared error could not recognize 5 of the 32 patterns (figure 3.6b).

Table 3.1: Comparison between Alopex and Backpropagation: Average number of iterations taken for convergence

	XOR	PARITY 3	SYMMETRY
ALOPEX (SQR. ERROR)	276	1874	7044
ALOPEX (SQRT. ERROR)	281	1860	6063
ALOPEX (LOG. ERROR)	108	792	4662
BACK PROP. (SQR. ERROR)	1571	5906	542
BACK PROP. (with momentum)	161	582	352

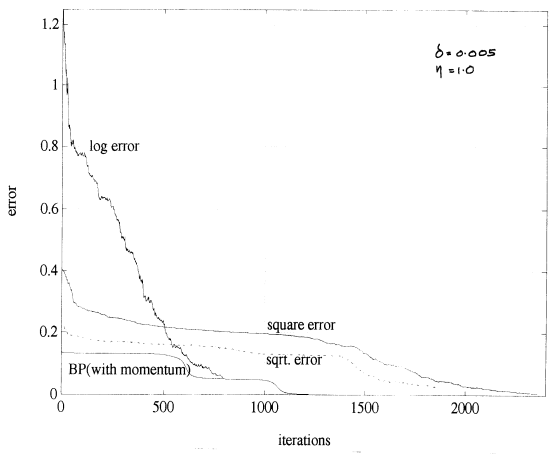


Figure 3.4 Convergence of the output error with Alopex and the backpropagation for parity−3 problem

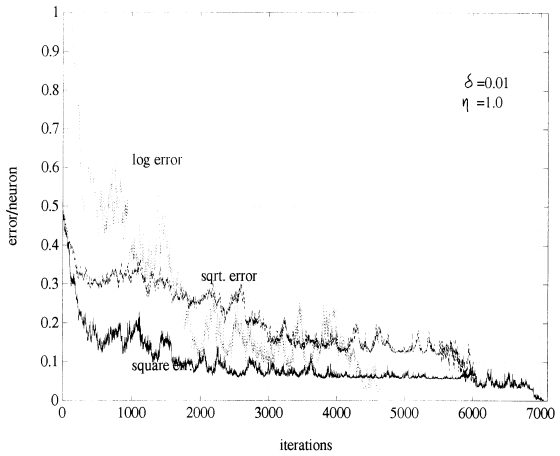


Figure 3.5 Convergence of the output error with Alopex algorithm for the symmetry problem

3.5.2. Symmetry Problem:

The vertical, horizontal and diagonal symmetries in a 4x4 binary pattern were presented to the network. There were 18 training patterns, 4 corresponding to vertical symmetry, 4 corresponding to horizontal symmetry and 5 each for two diagonal symmetry. For the studies with Alopex, a stepsize of 0.01 and a learning rate of 1.0 were used. A learning rate of 0.7 and

teristics. It could be seen that the number of iterations taken by Alopex is far less than that for BP, without the momentum term. It is comparable even with the version of BP which contains a momentum term. Figure 3.2 shows the convergence of the network error in a typical trial, for the XOR problem.

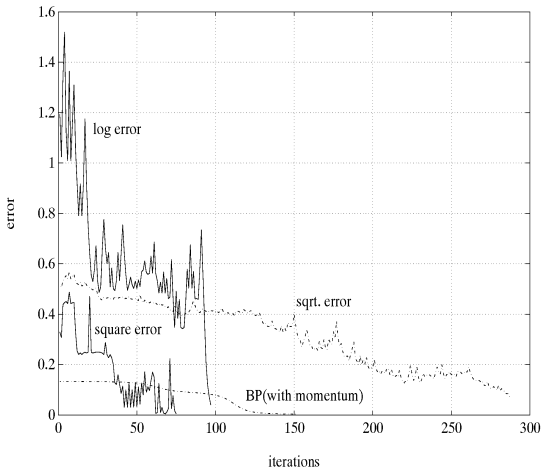


Figure 3.2 Convergence of the output error with Alopex algorithm using different error functions and backpropagation using square error (on XOR problem)

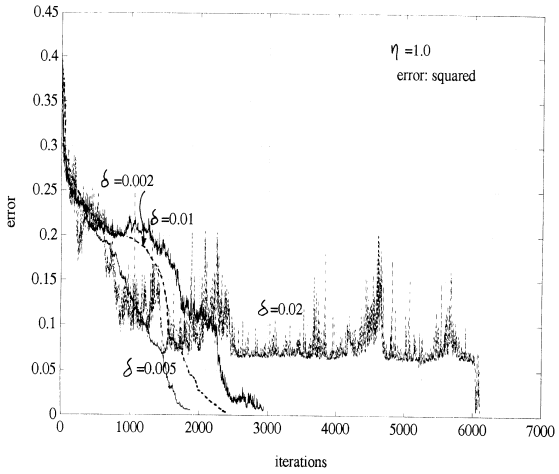


Figure 3.3 Effect of the step−size on the convergence of the output error;Parity−3 Problem. Note that a higher stepsize need not produce faster convergence beyond some value.

The results for parity–3 task are shown in table.3.1. The stepsize and learning rate for Alopex algorithm are 0.005 and 1.0 respectively, while the learning rate for BP is same as that for the XOR problem. As in the case of XOR problem, the log error clearly has a better convergence characteristics. The variation of error convergence as a function of the stepsize is shown in figure 3.3. A higher stepsize does not guarantee a faster convergence; in fact it may result in no convergence at all. On the other hand, a small stepsize results in an increase in the number of iterations needed for convergence. A stepsize 0.005 gave the best result for this problem. The convergence of the error at the output neuron is shown in figure 3.4.

the maximum error allowed at each of the output neurons for the convergence of the network, is taken as 0.1.

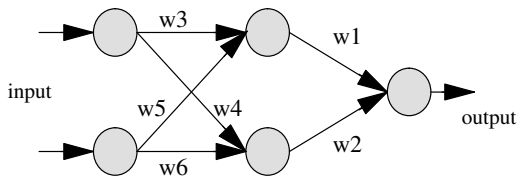


Figure 3.1 The 2−2−1 network architecture for XOR problem

3.5.1. Parity Learning:

In the parity problem, the desired output of the network is '1' if the number of 1s in the input pattern is odd and 0 otherwise. The complexity of the problem increases as the number of input bits is increased. The problem is nontrivial since most similar input patterns (those which differ by a single bit) has different target outputs and N number of hyperplanes are needed to solve a problem of dimension N. The first set of experiments were on the exclusive–or (XOR) problem, which is a parity problem of dimension two. A layered network architecture, as shown in figure 3.1, was used for the studies. In general, N hidden neurons were used in order to solve parity with patterns of length N.

The Alopex algorithm was tested on XOR problem with binary inputs '0' and '1'. Table 3.1 shows the average number of iterations taken in each case, over 25 trials with different initial weight distribution. The stepsize and learning rate chosen for the simulations using Alopex algorithm, are 0.01 and 1.0 respectively. The results using BP also are shown in table.3.1. The learning rate and momentum rate for BP are 0.7 and 0.9 respectively. It should be noted that these parameters are chosen over a number of trials to get the best convergence charac-

implicit manner and an appropriate annealing scheme could be chosen for faster convergence[50][58].

(e) Networks containing feedback between neurons from a single layer or from different layers can be used[50].

(f) Algorithm is a separate entity and does not depend upon the network architecture. For the same reason, it is ideally suited for digital VLSI implementation in a hierarchical single instruction multiple data (SIMD) architecture[87].

(h) Alopex is more plausible biologically than many of the other learning algorithms[38–45].

3.5. SIMULATION RESULTS FOR STATIC LEARNING TASKS

As mentioned before, the Alopex algorithm does not impose any restrictions on the nature of the error function. For specific classes of problems, smoother error surfaces, with fewer local minima, could be obtained by choosing an appropriate error measure. In order to demonstrate this fact, the Alopex algorithm was compared with BP algorithm using three different error functions, on the standard problems such as XOR, Parity–3 and Symmetry. Note that the original formulation of BP, could be used only with the squared error. The error functions used with the Alopex algorithm are:

(i) Squared error; $\sum (d - o)^2$

(ii) Square root error; $\sqrt{\sum (d - o)^2}$ and

(iii) Information theoretic error measure; $\sum d \log(d/o)$[76],

where d and o are the desired and actual outputs of the output neurons. In all the simulations,

the algorithm, the weight updating at some instants may be in the direction opposite to the gradient. In applications, such as signal prediction and tracking control, this may not be a desirable property as the updating in those cases are done normally based on the instantaneous error (note that there is no concept of discrete patterns here). One way to approach this problem is to update the weights at every N iterations rather than updating at each iteration. In this case, the squared error accumulated over N iterations could be expressed as

$$E = \frac{1}{2N}\sum_{0}^{N} \| y_d - y \|^2 \qquad (2.19)$$

This was found to be a necessary requirement when the Alopex algorithm was used for control applications, as discussed in Chapter 4 and 7.

3.4. IMPORTANT PROPERTIES OF THE ALOPEX ALGORITHM

Following are the important properties of the Alopex algorithm, as a learning algorithm for neural networks:

(a) The algorithm is synchronous; that is, all the weights in the network are updated synchronously. This allows an efficient parallelization in the hardware[87].

(b) Any error measure can be used. Appropriate error measures which give smoother error surfaces could be used for specific problems[50][55][58][61].

(c) The transfer functions of individual neurons need not be differentiable[58].

(d) The stochastic nature of the algorithm allows to overcome the local minima. The algorithm uses the simulated annealing[37] approach in an

3.3.1. Temperature Perturbation

In the expression (3.14) the value of $P_{ij}(n)$ becomes a step function when $t \rightarrow 0$ and a constant value 0.5 as $t \rightarrow \infty$. The choice of *'T'* is directly related to the feedback term. For larger values of *'T'*, the system produces random walks preventing the system from settling into any minimum. A smaller *'T'* causes the system to get trapped into a local minimum. This temperature term is equivalent to the one in the simulated annealing approach proposed by Kirkpatrick *et al.*[37]. As in the case of simulated annealing, the algorithm could be started with a high initial temperature which is decreased as the network converges.

An alternate way of escaping out of local minima, while keeping the auto–annealing scheme described before, is by 'perturbing' the temperature when the the change in error is very small. Let $T(n)$ and $T(n-1)$ be the 'temperatures' at instants n and $(n-1)$. Then the temperature at each instant could be modified by the following equation:

$$T(n) = \gamma \; T(n-1) \tag{2.17}$$

$$\text{if} \qquad E(n-N) - E(n) < \varepsilon \tag{2.18}$$

where γ is the 'perturbation rate' and ε is the desired change in error over N iterations. Choice of γ and ε may be problem dependent. Simulation studies have shown that typical values of γ may range from 1.05 to 1.1 for a decrease in error at the rate of 1% over 100 iterations.

3.3.2. Batch Updating:

It can be seen from equation (3.13) that the weight updating in the case of Alopex algorithm is done without calculating the error derivatives explicitly. Due to the stochastic nature of

over the previous two iterations(for the first two iterations, $P_{ij}(n)$ is taken as 0.5). As before, in the expression for $P_{ij}(n)$, T is a positive 'temperature' that determines the effective randomness in the system. With a non–zero value for T, the algorithm takes biased random walks in the direction of decreasing E. The simulations are started with a large value for the temperature T. Subsequently, the temperature is set equal to the average value of the correlation Δ_{ij}. That is,

$$T = \langle|\Delta|\rangle \tag{2.16}$$

This method reduces automatically the temperature when the parameters are close to the optima and hence the correlations are small. Towards the end, the step size also can be reduced for precise convergence.

The value $\langle|\Delta_{ij}|\rangle$ is computed by averaging Δ_{ij} over all the weights and over a reasonable number of iterations (say ten). The temperature T is then adjusted every tenth iteration. This causes gradual lowering of T, because of decreasing changes in E, as the minimum is approached. For the first iteration all W_{ij}'s can be chosen randomly and then total error $E(1)$ can be calculated. Then all the weights can be updated with a probability of 0.5, using equations (3.12) and (3.13), and the total error $E(2)$ can be calculated.

Note that the equations for updating the weights in the case of Alopex algorithm do not depend up on the functional form of the error and the neuronal transformation. The algorithm is a separate entity, independent of the network architecture. Relevant properties of the Alopex are discussed further in the later chapters. In this thesis we propose two subtle changes to the aforementioned algorithm, namely temperature perturbation and batch updating, in order to make it universally applicable to a wide variety of network architectures.

$$net_i = \Sigma w_{ij}\ out_j + \theta_i \tag{2.10}$$

where θ_i is the threshold of neuron i. For a neuron with a transformation $f(.)$ the output is then $out_i = f(net_i)$. The function f(.) could be of any form. For a sigmoidal transformation function with bounds (–1.0, 1.0) the output of the neuron could be given as:

$$out_i = \frac{1 - exp(- net_i / T_0)}{1 + exp(- net_i / T_0)} \tag{2.11}$$

where T_o is the gain of the sigmoid.

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

$$w_{ij}(n) = w_{ij}(n - 1) + \delta_{ij}(n) \tag{2.12}$$

where $\delta_{ij}(n)$ will have a small positive or negative step of size δ with the following probabilities:

$$\delta_{ij}(n) = \begin{cases} -\delta & with\ probability\ \ P_{ij}(n) \\ +\delta & with\ probability\ \ (1 - P_{ij}(n)) \end{cases} \tag{2.13}$$

The probability $P_{ij}(n)$ as in equation(3.7) is the Boltzmann distribution, namely,

$$P_{ij}(n) = \frac{1}{1 + exp(- \Delta_{ij}(n)/ T)} \tag{2.14}$$

where $\Delta_{ij}(n)$ is given by the correlation:

$$\Delta_{ij}(n) = \Delta w_{ij}(n)\Delta E(n) \tag{2.15}$$

$\Delta w_{ij}(n)$ and $\Delta E(n)$ are, respectively, the changes in the weight w_{ij} and the error measure E

where *P(n)* is the Boltzmann probability distribution given by:

$$P_i(n) = \frac{1}{1 + e^{-\frac{\Delta_i(n)}{T}}} \tag{2.7}$$

In equation (3.7), $\Delta_i(n)$ corresponds to the correlation between the change in the parameter $b_i(n)$ and the objective function $R(n)$ and T is the effective 'temperature' which determines the randomness of the system. That is,

$$\Delta_i(n) = [b_i(n-1) - b_i(n-2)][R(n-1) - R(n-2)] \tag{2.8}$$

Choice of 'temperature' T is related directly to the feedback (or correlation) term $\Delta_i(n)$. Simulation studies on a large class of problems showed that a subsidiary condition

$$\frac{\| \Delta(n) \|}{T} \simeq 1 \tag{2.9}$$

effectively allows the system to converge to a global optima.

3.3. THE ALOPEX ALGORITHM AS A LEARNING ALGORITHM FOR NEURAL NETWORKS

An alternate formulation of the Alopex algorithm is extensively used in the present studies. In this formulation[50][55–62][86], the learning in neural networks is treated as an optimization process. The objective is to minimize an error measure E, which is a function of the network weights $\{w_{ij}\}$ and training inputs. The algorithm can be described as follows:

Consider a neuron i with an interconnection strength w_{ij}, from neuron j. The summed output at the neuron i is,

$$I_j(n) = v(n)[b_j(n) + r_j(n)] \tag{2.2}$$

In equation (3.2) *v(n)* is a normalization factor selected such that

$$\sum_{j=1}^{N^2} I_j(n) = K, \qquad a\ constant \tag{2.3}$$

The cumulative bias $\boldsymbol{b}(n)$ are updated to maximize the response *R(n)*, by the following equation:

$$b_j(n) = b_j(n-1) + \eta[R(n-1) - R(n-2)][I_j(n-1) - I_j(n-2)] \tag{2.4}$$

where η is the updating rate. For the first two iterations, all the bias values are taken identical.

3.2. THE ALOPEX ALGORITHM AS AN OPTIMIZATION ALGORITHM

The Alopex process described by equations (3.1) to (3.4) was used, in various forms, in a number of later studies on the role of feedback in visual perception[42][44]. It was later extended as an optimization algorithm by Pandya[40] and was studied on a variety of problems [38][39]. Some of these problems include the diet problem[40], the problem of curve fitting and the travelling salesman problem. In the original formulation, the algorithm avoids the local minima by making use the noise term *r(n)*. It was later replaced by an effective temperature term (*T* in equation (3.7)). The Alopex process which was extended to an optimization algorithm is as follows:

If $b_{i=1...N}(n)$ represents the *N* parameters which are to be optimized, the updating equations are as follows:

$$b_i(n) = b_i(n-1) + \delta(n) \tag{2.5}$$

$$\delta(n) = \begin{cases} +\delta & with\ probability\ P(n) \\ -\delta & with\ probability\ (1 - P(n)) \end{cases} \tag{2.6}$$

second set consists of two problems. The first one refers to recognizing undersea targets from side–scan sonar returns, in which the network has to learn the spectral temporal dependencies of the consecutive input patterns. A time–delay network architecture[47][48] is employed for the recognition. These ideas are further extended in the studies on the second problem viz., the learning of temporal XOR signals[49]. A recurrent network architecture employed in these studies shows the effectiveness of Alopex algorithm on networks with feedback connections. Note that many of the popular learning algorithms, such as backpropagation, cannot be used directly on recurrent networks and are computationally intensive for such tasks.

3.1. THE ALOPEX PROCESS

The original Alopex algorithm was proposed by Harth and Tzanakou[43], in 1974, for the experimental determination of the receptive fields of single cells in the visual pathway. It was further used in the simulation studies where visual perception was suggested as a dynamical process, rather than a unidirectional information flow[41]. In the original formulation, the application of the algorithm was to find the visual pattern (an array of light intensities) which maximizes the response from the individual neurons in the brain. That is, if

$$\boldsymbol{I}(n) = [I_1(n),\ I_2(n), \ldots . I_j(n) \ldots . I_{N^2}(n)] \qquad (2.1)$$

represents the NxN elements of a visual field during the n^{th} stimulus pattern, then the objective is to maximize the neuron response $\boldsymbol{R}(n)$ which is a function of $\boldsymbol{I}(n)$. The intensity $I_j(n)$ in equation (3.1) is made–up of a random contribution $r_j(n)$ and the cumulative bias $b_j(n)$. That is,

CHAPTER 2.

THE ALOPEX ALGORITHM FOR SUPERVISED LEARNING IN NEURAL NETWORKS

In this chapter, the Alopex algorithm is shown as an efficient supervised learning algorithm for neural networks[50][58][61]. A brief review of the original formulation of the algorithm [41][43] and the later version as an optimization algorithm[38][40], also are presented. Simulation studies were conducted on two different classes of problems: (i) Learning and classification of static input patterns; and, (ii) Learning and classification of dynamic input patterns. The algorithm is applied to two practical problems, pertaining to each from two aforesaid categories.

The first set of problems consists of the standard benchmark problems such as Parity, Encoder, Symmetry and the MONKS problems. Relevant to each of these problems, the algorithm is compared with the backpropagation method. The ability of the Alopex algorithm in practical classification tasks is demonstrated using a handwritten digit recognition problem. The

For: venu
Printed on: Sat, Feb 5, 1994 15:58:50

Document: Ch__3
Last saved on: Sat, Feb 5, 1994 15:42:47