The application of a massively parallel learning network to the passive detection of a sonar target (viz., a surface ship) is discussed in this chapter. The task comprises the following activities.

(a) Recording of the sea noise with and without the target ship in the vicinity of the recording platform.

(b) Preprocessing of the recorded data to generate the network training patterns.

(c) Design of the neural network.

(d) Selection of a learning algorithm.

(e) Generation of the simulation software for the neural network.

(f) Teaching the network using the training patterns and the selected algorithm.

(g) Generation of test patterns using steps (a) and (b) for different targets in different geographical locations.

(h) Testing the target detection capability of the network by presenting the test patterns to it.

These processes are pictured as a block diagram in Fig 5.1.

5.1 Recording of Sea Noise

For recording of the sea noise, the "SUBER" Remote-Controlled Acoustic Data Acquisition Module was used. The
Fig. 5.1 Neural network scheme for sonar target detection
electrical signal from the hydrophones corresponding to the underwater sound they picked up is fed to the instrumentation tape recorder after amplification by the precision conditioning amplifier (PCA).

Recordings with and without target ships sailing in the vicinity were taken at different geographical locations using the data acquisition modules deployed at these locations. The set-up for recording of the sea noise is shown in Fig. 5.2.

5.2 Preprocessing of the Data

The front-end processing is an essential element to any neural network technique. If given non-representative or inadequate training data, any neural network paradigm will perform poorly. Neural networks do provide a novel method of abstracting feature information into a distributed encoding. They do not, however, by-pass the critical stage in any pattern recognition task of adequately defining the salient and characteristic features of the data.

The preprocessing employed in the present work relies on the extraction of the spectral data using a Fast Fourier Transform (FFT) computation on the A/D converted signal obtained from the sea noise recordings. This technique shows the clustering properties of the spectral components better than more conventional coding methods and thus provides a more useful representation on which the classifier can train. It is also a
Fig. 5.2 Set-up for sea noise recording
fast, reliable and well-supported technique [6].

The analog output signal from the tape recorder corresponding to the recorded data was low-pass filtered up to a cutoff frequency of 750 Hz and subsequently amplified by 20 dB.

The analog to digital converter (A/DC) resolution was 12 bit and a sampling frequency of 2 kHz was selected since the prominent frequency components in the radiated noise of the target are expected only up to 1000 Hz.

A 1K-point FFT was computed on the digitized time-domain data samples from the A/DC to convert them into frequency domain samples which correspond to the spectral components in the signal. A twenty-point vector was formed from these spectral components in the selected bandwidth of the signal by segregating them into twenty consecutive frequency bins with twenty Fourier coefficients in each bin. Since each point in the vector was formed by adding up the squared coefficients in each bin, its value gives a measure of the energy contained in the respective bin. The above vector, after normalization, was presented as the input to the neural network.

Input vectors were generated from time-strips of sea noise recordings with and without targets, made at different geographical locations. Recordings made at one of these locations were used to generate the training vector patterns while those made at other locations were used to generate the testing ones.
5.3 Neural Network Design

A multi-layer perceptron was used for the experimentation. In the light of the discussion in section 3.4.3, a three-layer network configuration with the number of neurons in the middle layer being more (viz. exceeding by two) than three times that in the input layer was selected. The type of preprocessing proposed in section 6.2 needs 20 neurons in the input layer. Since separate neurons were assigned for "target present" and "target absent" outputs, a network with 20, 62 and 2 neurons respectively for the input, middle and the output layer resulted. This is designated as a 20 - 62 - 2 network.

The performance of two more network topologies viz. 20 - 62 - 1 and 20 - 11 - 2 also was studied for comparison purposes.

The above three networks are shown in Fig 5.3, 5.4 and 5.5 respectively.

5.4 The Learning Algorithm

The backpropagation algorithm was used since it is most suited for multi-layered networks. Using the momentum method and the exponential smoothing method, networks were separately trained and their performance was compared.

For every pattern vector presented, each neuron
Fig. 5.3 The 20-62-2 Neural network
Fig. 5.4 The 20-62-1 Neural network
Fig. 5.5 The 20-11-2 Neural network
computes the net value obtained from equn. (3.1) modified by a bias as mentioned in section 3.4.1. The output from the neuron was obtained by thresholding the above with the sigmoidal nonlinearity (ref. equn. (3.3) with a=1).

Depending upon the learning algorithm chosen, either equn. (3.20) or (3.21) was used to modify the weights associated with the neurons in various layers. The weight error derivative $\delta_j$ in these expressions was calculated using equn. (3.15) and (3.17) respectively for neurons in the output layer and those in all the other layers.

5.5 Simulation Software Structure

Though neural computing is basically a parallel distributed processing procedure, the mathematical operations involved in it can easily be carried out on conventional computers. This facility for computer simulation of neural networks offers phenomenal possibilities for experimentation.

The software that was designed for the simulation work under discussion suits multi-layered feed-forward networks with any topology (limited by the memory available in the computer), any learning algorithm and any number of training/testing patterns.

The network topology is characterised by the following quantities which are to be specified in the
network specification file.

(a) Total number of neurons in the network (including the fan-out neurons at the network input).

(b) Number of input neurons.

(c) Number of output neurons.

(d) First weight to last weight associated with each of the neurons.

In the network learning phase, all the input training patterns were specified into the software as files with details as to the number of files, file size and the total number of patterns. Output patterns (ie. target vectors) were specified as an array. While passing through this phase, the network gradually converges and its weights and biases attain their generalised values at the end of this phase.

In the network testing phase, patterns with features (to the extent of the information as to whether target is present or absent) unknown to the network were applied to it one after the other and the corresponding output response was compared with the correct feature.

5.5.1 The Learning Phase

This comprised the following steps:

(a) Specify the network topology, training pattern details and the values of the appropriate coefficients used in the learning algorithm.
(b) Initialize all weights, biases and their derivatives to zero.
(c) Specify the number of iterations to be carried out and a
starting value & the desired minimum value for the sum of
square error.
(d) Either read the weights and biases already available in a
file ("IPFILE") or generate them with random values. Also,
store them in a file ("STARTWTS") in the latter case.
(e) Read all the training patterns viz. the input and the
corresponding target patterns from the relevant files and the
target array respectively.
(f) Set the control appropriately so that the training patterns
are presented to the network one after the other in the same
order in which they were read or in a random order.
(g) Carry out one epoch of training. This includes the following
steps:
   (i) Apply one training pattern to the network.
   (ii) Compute the output of the network corresponding to the
applied input pattern and the weights & biases already
set in.
   (iii) Compute the error between the applied target value in
step(i) and the value obtained in step (ii).
   (iv) Using the above error, calculate the error derivatives
for all the weights and biases of the network.
   (v) Using the above derivatives in the learning algorithm,
modify all weights and biases. Replace the earlier
weights and biases by their respective modified values.
   (vi) Repeat steps (i) through (v) for the remaining patterns
to complete one epoch of training and this is taken as
one iteration. The weights and biases obtained at the end of every epoch serve as the starting weight and bias values for the subsequent epoch.

(h) Square the error values obtained in step (g)(iii) for one iteration and add them up to get the sum of square error. Store this value and the corresponding order of iteration in a file ("TSSFILE").

(i) If the sum of square error newly obtained in step (h) is less than its earlier value, update the latter to the new value and store the corresponding weights and biases obtained as per step (g)(vi) at the end of the relevant iteration in a file ("FINALWTS").

(j) Repeat steps (g) through (i) until the sum of square error becomes equal to or less than the minimum value specified in step (c) or the execution of the number of iterations specified therein, whichever happens earlier.

(k) Store the weights and biases obtained at the end of the last iteration in a file ("LASTWTS").

The software architecture for network learning is depicted in Fig. 5.6 through Fig. 5.10.

5.5.2 The Testing Phase

During the learning phase, the network traverses through a gradient descent path in the error surface and at the end of this phase involving sufficient number of iterations, it converges and settles at the global minimum. With its generalized weights and biases, which correspond to the features
Neural network
Simulation software (LEARNING PHASE)

- no. of neurons
- no. of inputs & outputs
- no. of patterns

Initialisations
Geneorder
Backpropagation
SSE error computation

Final weights storage
FINALWTS file

Fig. 5.6 Software architecture diagram 1
Fig 5.7 Software architecture diagram 2
Fig. 5.8 Software architecture diagram 3
Fig. 5.9 Software architecture diagram 4
Fig. 5.10 Software architecture diagram 5
latent in the training patterns, the network can now detect the presence of these features in the unknown patterns presented to it. This process is carried out in the testing phase which is a non-iterative, one-shot, forward-pass computation.

This phase involved the following steps:
(a) Specify the network configuration in the manner mentioned at the beginning of section 5.5.
(b) Specify all the input patterns for testing (comprising only input vectors and no target vectors) as done earlier for the training patterns.
(c) Read all the test patterns.
(d) Load in to the network, the generalized weights and biases from a file ("USEDWTS") which points to the contents of the file "FINALWTS" which was created during the learning phase.
(e) Set one test pattern at the network input.
(f) Compute the network output.
(g) Classify the network output with "target present" or "target absent" labels by applying appropriate thresholds to these outputs.
(h) Repeat steps (e) through (g) for all test patterns.

The software structure for network testing is given in Fig. 5.11.

The source programme for the neural network simulation was developed in C language and it was run on CYBER 180/830 mainframe computer.
Fig. 5.11 Software architecture diagram 6
5.6 Classification Labeling

Since ambient noise as well as target-radiated noise was used for both training and testing the network and as the problem is of two-class nature, four combinations are possible for the above. These were segregated under three classification labels as shown in Table 5.1 below. Out of the total testing patterns, the percentage of correct classification obtained under these labels for each network-algorithm combination was used to make a comparative study of their classification efficiency.

<table>
<thead>
<tr>
<th>Test Input</th>
<th>Classification Result</th>
<th>Classification Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient noise</td>
<td>Target Present</td>
<td>False Alarm</td>
</tr>
<tr>
<td>Radiated noise</td>
<td>Target Absent</td>
<td>Miss</td>
</tr>
<tr>
<td>Ambient noise</td>
<td>Target Absent</td>
<td>Correct Classification</td>
</tr>
<tr>
<td>Radiated noise</td>
<td>Target Present</td>
<td>Correct Classification</td>
</tr>
</tbody>
</table>

5.7 Conclusion

The preparation of data for presentation to the neural network, the design details of the network and the simulation software structure were discussed here. The test results are summarized and discussed in the chapter to follow.