NEURAL NETWORKS FOR DATA FUSION

By

FENGZHEN WANG, B. Eng., M. S. Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Ph.D. of Engineering

McMaster University

Extended Summary for **Neural Networks for Data Fusion**

From Fengzhen Wang

March 6, 1997

This thesis describes the results of research carried out on the use of neural networks for implementing data fusion.

The process of data fusion may be viewed as a multi-level hierarchical inference process whose ultimate goal is to assess a mission situation and identify, localize and analyze threats. Each succeeding level of data fusion processing deals with increasing levels of abstraction. The lower level is concerned solely with individual objects where sensor data are processed to derive the best estimates of current and future positions for each hypothesized object as well as inference as to the identity and key attributes of the objects.

A unique contribution in the thesis is that a new technique of measurement data association in a multitarget radar environment is developed. The technique is based on the mean-field-theory machine and has the advantages of both the Hopfield network and the Boltzmann machine. The new energy function built in the thesis can be considered as a cost function. the cost function takes a minimal value when the plot-track association is in the optimal status. In the technical development, three new energy functions have been created. Theoretically, the critical annealing temperature is found to determine the annealing temperature range. Neural data association capacities have been evaluated using data sets with and without clutter for different measurement radar precisions.

lished herein provides an affirmative assertion that the mean-field-theory machine globally asymptotically converges to a stable equilibrium provided the network satisfies the conditions described in the theorem. It also gives us a rule for devising a physical energy function. This is a remarkable achievement which helps to fill in a void in mean-field-theory machine.

As a benchmark, the nearest neighbour association is compared with neural data association. The neural data association has higher capacity than does the nearest neighbour association: it demonstrates remarkable improvement in high target density circumstances.

The new energy functions derived in the thesis have been extended to multiple dimensional data association. A comprehensive analysis carried out using computer simulations demonstrates that the new technique developed in the thesis possesses high association capacity in the presence of false alarms; as well, it can cope with track-crossing in dense target environments. In the case $\sigma_p = 0.3$, the neural data association capacity with multiple dimensional measurements is 30% higher than that only using position measurements for the environment described in Chapter 5.

Another unique contribution of thesis is the feature-mapping data fusion, the accuracy of which approaches the Cramer-Rao lower bound. The feature-mapping data fusion has many advantages over the conventional fusion approach. Since it is based on a neural network technique, it can learn from the observations and get the same precise fusion centroid as that obtained by applying ML in the case of Gaussian distributed measurement error. As we know, in the case of ML, the measurement error distribution function and sensors accuracies need to be known a priori. If the sensors accuracies are not available, the calculation overload is huge. If the error distribution function is unknown, ML is unable to contribute to the of ML. Feature-mapping data fusion can learn from any error distribution function, and fuse the observed data together. On the other hand, the method presented here, can be easily built into a VLSI network which possesses parallel computation potency. In other methods, as well, the cluster boundary should be found in order that the data, which are from the same target can be fused. In a dense target environment, the feature-mapping fusion approach avoids the difficulty in finding the cluster boundary by introducing adaptive lateral feedback in the postsynaptic layer. This makes the fusion implementation very feasible and easy.

In this thesis, we have studied neural networks and multisensor data fusion, and developed the techniques for multisensor target classification. Multilayer perceptron neural networks trained by Backpropagation have been designed. A feature set contaminated by noise. which possesses the dominant characteristics of targets and has a certain dynamic range, is chosen to train the networks. The entire system consists of identification (IN) and classification nets (CN). Each identification network is used to extract a particular feature of the target, then the outputs of the identification networks are fused by a classification network to give the decision. Like hierarchical structure of adaptive expert networks, the system structure makes the learning process fast. In the system, each subsystem is a simple neural network. It is easy to train the neural networks that have simple structures. For a simple target signature. one hidden layer is enough to make the simple network converge. If uncorrelated signatures are mixed together during the learning phase of a simple network, the network will not converge. Even though we can increase the number of hidden layers to make the network converge. however. the convergent speed is very slow due to the uncorrelated signatures. In a practical application of the back-propagation algorithm, learning results from the many complete presentation of the entire training set during the learning process is called an *epoch*. The learning process is maintained on an epoch-by-epoch basis until the synaptic weights and threshold levels of the network stabilize and the average squared error over the entire training set converges to some minimum value. In order to speed up the training or decrease the number of epoches used in the learning process, both momentum and adaptive learning rate (feasible in practice) methods are used. The simulation results to be presented in this thesis show that the technique of automatic target classification using neural networks can achieve robust decision performance.

Neural networks for data fusion have proven to provide for a flexible solution. It is easy to extend them to accommodate future changes. More importantly, without modifying the general structures of the networks, additional training facts can be added to the training sets so as to include more target features or perform more complex tasks. If more signatures are used to carry out target classification, we can deploy additional identification networks and increase the number of input nodes in the classification networks. If more than two targets are to be classified, the functions of the classification network can be extended by increasing the number of output neurons. In addition, once neural networks are well trained, real-time data fusion can be realized since neural network operation is a parallel one, which means that neural networks have prominent advantages over other devices.

Finally, the research results are summarized as follows:

- neural techniques for multisensor target classification,
- new technique of measurement data association,
- establishment of the convergent theorem for mean-field-theory machine,

- multidimensional data association,
- evaluation of data association performance,
- feature-mapping data fusion, and comparison with maximum likelihood fusion.

The attainments made in the thesis provide the approaches to solve the intractable data fusion problems.

NEURAL NETWORKS FOR DATA FUSION

PH.D. OF ENGINEERING (1997) MCMASTER UNIVERSITY

(Electrical and Computer Engineering) Hamilton, Ontario

TITLE:	Neural Networks for Data Fusion
AUTHOR:	Fengzhen Wang B. Eng. (Nanjing University of Science and Technology) M. S. Eng. (Nanjing University of Science and Technology)
SUPERVISOR(S):	Dr. John Litva, Professor Department of Electrical and Computer Engineering
NUMBER OF PAGES:	xx, 183

Abstract

The process of data fusion may be viewed as a multi-level hierarchical inference process whose ultimate goal is to assess a mission situation and identify, localize and analyze threats. Each succeeding level of data fusion processing deals with a greater level of abstraction. The lower level is concerned solely with individual objects where sensor data are processed to derive the best estimates of current and future positions for each hypothesized object as well as provide an inference as to the identity and key attributes of the objects. With the recent proliferation and increasing sophistication of new technologies, it is recognized that the incorporation of new techniques, such as neural networks and others, will make the data fusion system more powerful in tri-service (command, control and communications).

In this thesis, optimization neural networks are investigated. A new technique of measurement data association in a multitarget radar environment is developed. The technique is based on the mean-field-theory machine and has the advantages of both the Hopfield network and the Boltzmann machine. In the technical development, three new energy functions have been created. Theoretically, the critical annealing temperature is found to determine the annealing temperature range. A convergence theorem for the mean-field-theory machine is put forward. Based on the technique, neural data association capacities have been evaluated in cases with and without clutter, taking into account different accuracies for radar measurements.

New energy functions have been extended to multiple dimensional data association. A comprehensive analysis by computer simulations has demonstrated that the new technique developed here possesses high association capacity in the presence of false alarms; it can cope with track-crossing in a dense target environment.

A feature-mapping neural network for centralized data fusion is presented, and its performance is compared with that of the Maximum Likelihood approach.

In support of our study of multisensor data fusion for airborne target classification with artificial neural networks (ANNs), we designed a neural classifier. Multilayer perceptrons neural networks trained by back-propagation (BP) rule are discussed. In order to speed up the training or decrease the number of epoches in the learning process, both momentum and adaptive learning rate methods are used. The simulation results show that the technique of automatic target classification using neural networks has the potential to classify targets.

Acknowledgements

I am greatly indebted to my supervisor, John Litva, for initially allowing me the freedom to explore various research areas, and subsequently introducing me to the exciting world of data fusion and neural networks and providing invaluable support, encouragement, and recommendations as my work progressed. I would like to thank Defence Research Establishment Valcartier for its financial and technical supports. Special thanks must go to Eloi Bosse for the useful discussions. I am grateful to my supervising committee: Simon Haykin and Patrick Yip, for their suggestions and recommendations.

I greatly appreciate the advice, help, and companionship from all the past and present researchers and graduates I have had the pleasure of meeting. The last word must go to my family, for without my family's constant encouragement and support I would not be where I am today.

List of Symbols

$W(K)$	Kalman gain.
$\hat{x}(k\|k)$	State estimate at time k.
$\nu_j(k)$	Difference between the jth measurement and track at time k.
$\beta_j(k)$	Probability of the jth measurement coming from target.
$P(k\|k)$	Covariance matrix of the state $\hat{x}$.
P_D	Probability of detecting the target.
P_G	Probability of the target's measurement falling within the track's extension gate.
$S(k)$	Covariance matrix of $\nu_j(k)$.
o	Number of clutter measurements.
V	Volume of the extension gate of a track.
$\varphi(.)$	Limiting or threshold function; sigmoid activation function.
w_{ji}	Synaptic weights of neural network.
I_i	External excitation of neuron i.
x_i	Input of neuron.

y_j	Output of neuron.
γ	Positive constant or variable which controls the slope of the sigmoidal function.
I_{ji}	Current flowing through the resistor R_{ji}.
a_{ji}	Excitatory synaptic weight.
b_{ji}	Inhibitory synapse.
R	Resistor.
G	Conductance.
C_j	Capacitance.
I_j^e	External current applied to the input of neuron j in Hopfield net.
r	Scaling resistor.
$\mathbf{W}$	Matrix of synaptic weights.
$\mathbf{I}$	$\mathbf{I} = [I_1, I_2, ..., I_n]^T$.
φ_j^{-1}	Inverse of function of φ_j.
E	Energy function of neural network.
$x_j^{(k+1)}$	jth neuron output at $(k+1)$ step.
$\mathbf{x}$	$\mathbf{x} = [x_1, x_2, ..., x_n]^T$.
N_r	Random number between zero and one.
T	Annealing temperature.
N_i	Thermal noise component.
T_c	Critical temperature.

E^{opt}	Objective function to be minimized.
v	Output of analog neuron.
T_{ij}	Connection weight on the link from the *jth* neuron to the *ith* neuron.
d_{XY}	Distance between X position and Y position.
d_p	Penalty constant.
E_{Xi}	Mean field of a neuron.
v_{Xi}	Neuron output corresponding to plot X and track i.
α	Cooling constant; momentum constant.
$\overline{\mathbf{x}}$	Steady state of neural network.
Z_X	$n-$dimensional measurements.
P_j	*jth* predicted point.
σ_R	RCS measurement precision.
σ_p	Position measurement precision.
σ_v	Speed measurement precision.
L_{min}	Minimal track distance.
ρ	$\rho = \frac{L_{min}(\text{minimal distance})}{\sigma(\text{precision})}$.
$d_j(n)$	Desired response of the *jth* neuron.
$\epsilon_j(n)$	Error signal at the output of neuron j at iteration n.
$\zeta(n)$	Summation of squared error.
ζ_{av}	Average squared error.
θ_j	Threshold of neuron j.

η	Learning rate.
$\delta_j(n)$	Local gradient of neuron j.
$\mathbf{w}^*$	Optimal weight.
W_j	Weight vector.
$i(X)$	Best match neuron index.
$g(y)$	Positive scalar function.
$\Lambda_{i(X)}$	Topological neighbourhood of the winning neuron $i(X)$.
$\bigcap_{ji}$	Neighbourhood function.
$\sigma(n)$	Effective width of the topological neighbourhood.
$p_X(X/\Phi)$	Conditional joint probabilistic density function.
$l(\Phi)$	Likelihood function of Φ.
$L(\Phi)$	$L(\Phi) = \ln[l(\Phi)]$.
a_j	Target centroid.

List of Acronyms

ANN	Artificial neural network.
AR	Auto-regressive model.
BIBO	Bounded input–bounded output.
BP	Back propagation.
CN	Classification net.
FMDF	Feature mapping data fusion.
ID	Identification.
IN	Identification net.
JPDA	Joint PDA.
KCL	Kirchhoff's current law.
LMS	Least-mean-square.
LP	Linear programming.
MDA	Measurement data association.
MFT	Mean-field-theory.
MFA	Mean-field annealing.
MFTDA	Mean-field-theory data association.

ML	Maximum likelihood.
MTI	Moving target indicator.
MTT	Multiple-target-tracking.
NDA	Neural data association.
NPDA	Neural network PDA.
PDA	Probabilistic data association.
PD	Pulse Doppler.
PDF	Probabilistic density function.
PPI	Plan position indicator.
RCS	Radar cross-section.
SOFM	Self-organizing feature mapping.
TSP	Travelling salesman problem.
VLSI	very-large-scale-integrated circuit.

Contents

List of Figures

Chapter 1

Introduction

1.1 Survey of Data Fusion

Data fusion consists of a data integration or correlation process which combines data from multiple sensors to derive meaningful information not available from any individual sensor[1]. The process of combining data has been variously referred to as multisensor or multisource correlation, multisource integration, or sensor blending [1, 2]. It is a multilevel process (i) dealing with detection, association, and estimation: it combines data and information from multiple sensors to achieve refined state and identity estimation, and (ii) inferring target intent and assessing situations.

The data fusion function continuously transforms data and information from multiple sources into richer information concerning: individual objects and events; current and potential future situations; and vulnerabilities and opportunities to friendly, enemy and neutral forces. In these contexts, data fusion provides continuous "refinement" of input data from a variety of sources in support of decision requirements or "assessments" which may be required at any time by operators and commanders. The process of data fusion may be viewed as a multi-level hierarchical inference process whose ultimate goal is to assess a mission situation and identify, localize and analyze threats. Each succeeding level of data fusion processing deals with a greater level of abstraction. The lower level is concerned solely with individual objects where sensor data are processed to derive the best estimates of current and future positions for each hypothesized object as well as inference as to the identities and key attributes of the objects.

Situation assessment deals with the continuous inference of statements about the tactical picture provided by the lower level data fusion function. It is concerned with associating hypothesized objects with known and expected organizations and events, and it uses what is known about enemy doctrine and objectives to predict the strengths and vulnerabilities for threat forces and friendly forces.

Situation assessment, together with operator interaction, is used to derive planning and decision support functions for allocating and scheduling the use of critical defence resources and coordinating defence actions in support of the mission.

The data fusion has the following characteristic features:[1]

1. robust operational performance,

2. extended spatial coverage,

3. extended temporal coverage,

4. increased confidence,

5. reduced ambiguity,

6. enhanced spatial resolution,

7. improved detection performance,

8. improved system operational reliability,

9. increased dimensionality.

Since it has encouraging advantages, multisensor data fusion increasingly draws the interest and attention of researchers in a number of relevant fields. Research activities and results in directly and indirectly related areas are reported annually at the US tri-service data fusion symposium held at John Hopkins University and at a variety of the annual SPIE conference, and IEEE Transactions, among others. A number of promising theoretical developments and technological branches are:

1. development of optimum decentralized detection, estimation, and tracking methods,

2. development of representation and management calculi for reasoning in the presence of uncertainty,

3. development of spatial and contextual reasoning processes suitable for assessment of activity and intent, based on temporal and spatial behavior,

4. development of integrated, smart sensors with soft-decision signal processing for use in systems that perform numerical and symbolic reasoning in uncertainty,

5. development of both distributed and integrated data processing architectures applicable to data fusion,

6. decomposition of processing algorithms into parallel processes for implementation on parallel machines,

7. creation of neural networks and application of neural network technology to detection, tracking, classification, and assessment problems.

As mentioned above, data fusion is a multi-level process. Level 1 data fusion is defined as the first stage of a process which deals with the detection, association, and estimation: it combines data and information from multiple sensors to achieve refined states and identity estimation. For level 1 data fusion, there are three main functional models: *data association, data estimation, and identity declaration*. The functional role for *Data association* is based on the proposition that given n observations from one or more sensors, how do we determine which observations belong together, and which identities among observations are of the same entity, in which there are three subdivisions to be considered, observation to observation, observation to track, and observation to observation for identity. In the course of data association, alignment, prediction (performed in data estimation), and gating have to be implemented before assignment. Alignment functions in a multisensor scenario: it transforms the measurements into the same reference coordinates. Gating performs an initial screening to eliminate unlikely observation-to-observation or observation-to-track. The assignment step invokes decision logic to declare that two observations or an observation and a candidate track belong together, i.e. they represent the same underlying physical entity or process.

The factors to be considered in data association include:

1. what metric will be used to measure the similarity between observations?

2. how many scans of the data will be used?

3. will an observation be uniquely assigned to another observation or track?

4. how many and what kind of hypotheses will be considered at any one time?

Data estimation deals with the problems related to (i) determining the location of a stationary entity, (ii) determining the position and velocity of a moving object (in tracking), (iii) estimating the inherent characteristics of an entity, (iv) estimating the parameters of a model. The conventional optimization criterion used in estimation is one of the following: least squares, weighted least squares, minimum mean squared error, Bayesian weighted least squares, or maximum likelihood.

Identity declaration uses pattern recognition techniques such as templating, cluster analysis, adaptive neural networks, or physical model techniques to recognize an entity's identity. The concise description of the pattern recognition technology is given as follows: In the parametric templates, preestablished parametric limits or boundaries are used to define identity classes. Cluster algorithms use heuristic methods to group data into the natural clusters which represent object identity. In the neural network technology, adaptive neural networks emulate biological nerve connections to perform a nonlinear transformation between an input feature vector and an output identity class. On the other hand, the physical model matches observed signature data against a predicted signature, computes correlation between the predicted and observed data.

The three functional models of data association, data estimation and identity declaration can be joined together to obtain extended reasoning. The output of each model may be used by other models as an input to improve the functions of these models for identifying the events with high capacity. One example consists of the case where data association is structured to partially use the target's attributes.

1.2 Review of Data Association

Often in multitarget tracking, there is more than one measurement available for updating the state of a single target. To solve the problem of data association between targets and measurements, two typical approaches have been reported in the literature in the 1970s. One is called the target-oriented approach in which each measurement is assumed to have originated from either a known target or clutter, as in probabilistic data association (PDA) [3, 4, 13, 14] and joint probabilistic data association (JPDA)

[3, 5, 13]. The other is called a measurement-oriented approach in which each measurement is hypothesized to have originated from either a known target, a new target, or clutter [6]. Recently, a new approach is proposed in [7] to deal with the data association problem in multitarget tracking. The new approach is called a track-oriented approach in which each track is hypothesized to be either undetected, terminated, associated with a measurement, or linked to the start of a maneuver. In these approaches, the number of data association hypotheses can increase rapidly with the increase in the number of targets and the number of measurements. Therefore, in the case of multitarget tracking algorithms, the computational cost of generating the data association hypotheses can become excessive when both the number of targets and the number of measurements are large. In [12], The unified framework was developed to provide a systematic scheme for the generation of data association hypotheses in the target-oriented, measurement-oriented, and track-oriented approaches. By adapting the depth-first search (DFS) algorithm, a huge amount of calculation can be alleviated. In PDA, the computational cost for data association is reduced drastically by isolating targets from each other. Only the measurements which lie in the validation gate of a target are considered for data association. Since the PDA ignores the interference from other targets, it may, sometimes, cause tracking errors in the case of closely spaced targets, as discussed in [5]. This difficulty is greatly reduced by using JPDA [5], because it takes into account the fact that if a measurement falls inside the intersection of the valid gates of several targets, it could have originated from any one of these targets or from clutter. In JPDA, targets are divided into clusters.

The targets are in the same cluster if there is at least a measurement inside each of the intersections of their valid gates. The computational cost for data association is reduced by grouping targets into clusters. The number of different data association hypotheses in each cluster is still an exponential function of the number of targets in the cluster. Due to the complexity of data association, the implementation of a multitarget tracking algorithm has only been carried out in a 2-target case [5. 6], except in [8], where JPDA filtering was used to track 11 nonmaneuvering and maneuvering targets.

In order to get a picture of the mechanism which supports PDA and JPDA. let us review them briefly. PDA is a method of associating measurements detected in the current readout of the sensor, or scan, with current tracks using a probabilistic score. A score is a measure of how well the measurement-to-track association fits. Once the score for each measurement is calculated, the tracks are updated with a weighted sum of the measurements, where the weights are the calculated probabilities. This has been considered to be a very good method for tracking targets in dense clutter without having the large processor and memory overheads that are needed for multiple hypothesis tracking. PDA can be extended to tracking maneuvering targets. A detailed derivation of the PDA can be found in [3] and is briefly described here. When using PDA or JPDA, the update equation of the Kalman filter becomes

$$\hat{x}(k|k) = \hat{x}(k|k-1) + W(k)\nu(k), \tag{1.1}$$

where $\hat{x}(k|k)$ is the state estimate at time k given k updates, $\hat{x}(k|k-1)$ is the prediction of the state at time k given $k-1$ updates, $W(k)$ is the Kalman gain and the weighted sum of the measurements takes on the form

$$\nu(k) = \sum_{j=1}^{m} \beta_j(k)\nu_j(k), \tag{1.2}$$

where $\nu_j(k)$ is the difference between the jth measurement and the predicted position of the track at time k. $\beta_j(k)$ is the probability that the jth measurement came from the current target in track. The updated covariance matrix is given by

$$P(k|k) = \beta_0(k)P(k|k-1) + [1-\beta_0(k)]P^c(k|k) + \tilde{P}(k), \tag{1.3}$$

where

$$\tilde{P}(k) = W(k)\left[\sum_{j=1}^{m} \beta_j(k)\nu_j(k)\nu_j^T(k) - \nu(k)\nu^T(k)\right] W^T(k), \tag{1.4}$$

$$P^c(k|k) = [I - W(k)H(k)]P(k|k-1), \tag{1.5}$$

where P is the covariance matrix of the state $\hat{x}$, H is the linearized mapping of the state onto the measurement space, I is the identity matrix, and β_0 is the probability that none of the measurements in the extension gate originate from the target in the track.

The probabilities of the jth measurement originating from the target in track at time k is given by

$$\beta_j(k) = \frac{e_j}{b + \sum_{l=1}^{m} e_l}, \qquad j = 1, ..., m,$$

$$\beta_0(k) = \frac{b}{b + \sum_{l=1}^{m} e_l}, \tag{1.6}$$

where

$$e_j = \exp[-1/2\nu_j^T(k)S^{-1}(k)\nu_j(k)], \tag{1.7}$$

$$b = \lambda|2\pi S(k)|^{1/2}(1 - P_D P_G)/P_D, \tag{1.8}$$

λ is the spatial sensitivity of false measurements, P_D is the probability of detecting the target, P_G is the probability of the target's measurement falling within the track's extension gate, and $S(k)$ is the covariance matrix of $\nu_j(k)$ which is computed in the Kalman filter gain equation. This calculation of the probabilities assumes that there is one measurement in the track's extension gate which originates from a target and the rest are random clutter which will average out to zero in the weighted sum of (1.2). If the measurement of another target is persistently in the extension gate of this particular target, the probability calculation will be wrong and poor tracking will result. To account for multiple targets, the JPDA was developed.

The JPDA is the same as the PDA with the exception of the probability calculations. The probabilities are computed as joint probabilities to account for the fact that the measurements under consideration may have come from more than one target. The probability of track 1 being associated with measurement j in the JPDA is computed from the law of total probability and is given [3]

$$\beta_j(k) = \sum_{\theta(k)} P\{\theta(k)|Z^k\}\omega(\theta(k)), \tag{1.9}$$

where $\omega(\theta(k))$ is a binary variable indicating whether joint event $\theta(k)$ contains the association of track 1 and measurement j. A joint event is a set of measurement-to-track associations which have measurements assigned to either clutter or to one track, and each track assigned to only one measurement or declared missed. Z^k denotes a cumulative set of measurements. The probability of the individual joint event is given by

$$P\{\theta(k)|Z^k\} = \frac{1}{c}\frac{\phi!}{V^{\phi}}\prod_{j=1}^{m}[N[\nu_j(k)]]^{\tau_j} \times \prod_{t=1}^{T}(P_D^t)^{\delta_t}(1-P_D^t)^{1-\delta_t}, \tag{1.10}$$

where ϕ is the number of clutter measurements, V is the volume of the extension gate of the track. m is the number of measurements on this scan, τ_j is a binary variable indicating whether the measurement is assigned to a track, T is the number of current tracks. P_D is the probability of detection, and δ_t is a binary variable indicating whether a track has been assigned to a measurement.

Calculating these joint probabilities is a formidable task. To find the probability that track 1 will be legitimately extended by adopting measurement i, the algorithm must calculate all feasible joint events, compute their probabilities using (1.10) and then add all probabilities from joint events which include the event of association of track 1 with measurement i. The computation of all of the joint events has been shown to increase computer loading exponentially as the number of targets increases.

To reduce significantly the computational cost for data association, several approximations to the JPDA and the algorithm in [6] have been reported in literature. A first version of the approximate or "cheap" JPDA was published in [9], where the probability of association of target t with measurement j was computed by an *ad hoc* formula. The "cheap" JPDA in [9] performed fairly well in the case of two targets but failed to track four targets [10]. In [15], a fast JPDA algorithm was proposed for the computation of the β_j^ts in the JPDA. For $j > 0$, $\beta_j^t(k)$ is the *a posteriori* probability that measurement j originates from target t at time k. For $j = 0$, $\beta_0^t(k)$ is the *a posteriori* probability that no measurement originates from target t at time k. In the fast JPDA algorithm, the computation of $\beta_j^t(k)$ was implemented using an enormous number of vector inner product operations. An optical processor was suggested in order to approximately realize these operations. Alternatively, Nagarajan, et al., [16] arranged the hypotheses in the measurement-oriented approach [6] in a special order so that the probabilities of the hypotheses are proportional to the product of certain probability factors that have already been evaluated. The algorithm locates the N globally best hypotheses without evaluating all of them.

One of the useful methods for data integration is the maximum likelihood, which is distinct from the PDA or JPDA in terms of its data model [19]. In some situations we can describe target-originated measurements as random variables, independent from scan to scan, with probability distributions which are given except for some unknown non-random parameters. Such target models are applicable to the sensor fusion problem with independent sensors. One example is the problem of location of several targets using returns from several sensors. The sensors may be of different types (bearing, range, velocity, etc.). Moving targets may also be included if their motion is described by a deterministic model with unknown parameters, and measurement noise is independent from scan to scan. One way to solve this kind of problem in such a situation is to perform a search for the measurement-target association which maximizes the likelihood of the observed measurements. This results in an optimal integration, but computationally it is costly. This is the trade off of using the maximum likelihood approach.

In [10] the JPDA was emulated by a neural network. However, a comprehensive analysis [11] reveals several drawbacks of the neural network approach developed in [10]. Sengupta and Iltis [10] formulated the computation of the *a posteriori* probabilities for the JPDA as a constrained minimization problem. This technique, which is based on the use of neural networks, was also extended so as to apply to maneuvering targets. By comparison with the travelling salesman problem (TSP), they proposed a Hopfield neural network to approximately compute the β_j^ts and called this neural network probability data association (NPDA). In fact, β_j^t is approximated by the output voltage V_j^t of a neuron in an $(m+1) \times n$ array of neurons, where m is the number of measurements and n is the number of targets. Sengupta and Iltis claimed that the performance of the NPDA was close to that of the JPDA in those situations where the numbers of measurements and targets were in the ranges of 3 to 20 and 2 to 6, respectively. There were six penalty constants in the energy function built by Sengupta and Iltis. Since there are no strict guidelines for choosing the constant coefficients of the energy function for the Hopfield neural networks, the coefficients used in the NPDA were selected essentially by trial-and-error. However, using the coefficients chosen by [10], the architecture of the neural network in [10] degenerates into individual columns with identical connections and the input currents throughout the neural network are almost of the same amplitude. Furthermore, the neural network in [10] has a strong tendency to converge to the V_j^ts which are close to $1/(m+1)$ with the given initial values for the V_j^ts. Zhou and Bose in [11] pointed out that it might be a good idea to use the neural network to approximately compute the *a posteriori* probability β_j^t in the JPDA. However, the neural network developed in [10] was shown to have improper energy functions. This resulted from misinterpretations of the properties of the JPDA which the network was supposed to emulate. Furthermore, improper choices of the constant coefficients in the energy function in [10] make the situation worse.

Data association can be viewed as an example of the classical assignment problem. The optimal solution to an assignment problem minimizes problem-oriented cost function. Smith et al. [17] built an energy function of Hopfield neural network for measurement data association. The energy function was a cost function which was related to the total distance between measurements (plots) and tracks. In order to obtain a stable solution, the authors used inhibition measurement technique and desired input computation technique. The inhibition measurement technique is based on measurements of the inhibition on a row by row and a column by column basis, variations to the sensitivity of the row/column inhibition, and diminution of this sensitivity as the network iterates. The desired input computation technique centers on starting out with large values to ensure stability, then gradually decreasing the desired input as the network iterates to generate only valid solutions. However, the energy function built by Smith has five constants to be chosen arbitrarily, and it often sticks to the local minimal point, which makes this method a shadowy one for applications. Another approach for solving the assignment problem is $0 - 1$ integer programming [18]. The problem is how to build a reasonable objective function.

1.3 Review of Classification

Computerized information extraction from sensor data sources has been studied over the last two decades. The data used in the processing have mostly been multispectral. and the statistical pattern recognition methods (multivariate classification) are now widely known. There may be many kinds of data available from different sensors observing the same scene. These are collectively called multisource data. It is interesting to use all these data to extract more information and get higher accuracy when carrying out the classification. However. the conventional multivariate classification methods can not be used satisfactorily in processing multisource data [20]. This is due to several reasons. One is that the multisource data can not be modeled by a convenient multivariate statistical model since the data are multitype. They can. for example. be spectral data. elevation ranges, and even non-numerical data. The data are also not necessarily expressed in common units, and therefore scaling problems may arise. Another problem with statistical classification methods is that the data sources may not be equally reliable. This means that the data sources need to be weighted according to their reliability, but most statistical classification methods do not have such a mechanism. This all implies that methods other than conventional multivariate classification must be used to classify multisource data.

Recently, there has been a great resurgence of research in neural networks. New and improved neural network models have been proposed for classifying complex data. The generalized delta rule is an example of such a method. Neural network models have such an advantage over statistical methods that they are distribution-free and no prior knowledge is needed about the statistical distributions of the classes in the data sources in order to apply these methods for classification. The neural network methods also take care of determining how much weight each data source should have in the classification. A set of weights describes the neural network, and these weights are computed using an iterative training procedure. On the other hand, neural network models can be very complex computationally, and the iterative procedures that are used for training are sometimes slow to converge. The performance of the neural network models in classification is dependent on representative training samples, whereas the statistical approaches need to have an appropriate model of each class.

1.4 Scope of Thesis

This thesis describes the results of research carried out on the use of neural networks for implementing level 1 data fusion.

Generally speaking, an artificial neural network (ANN) is an information or signal processing system composed of a large number of simple processing elements, called artificial neurons or simply nodes, which are interconnected by direct links called connections and which cooperate to perform parallel distributed processing in order to solve a desired computational task. One of the attractive features of ANNs is their capability to adapt themselves to special environmental conditions by changing their connection strength or structure. Artificial neural networks are sometimes considered as grossly simplified models of the human brain. This view is somewhat exaggerated and misleading because the human brain is not well understood and indeed its behaviour is very complex. In our opinion it is more reasonable to compare artificial neural network capabilities to simpler nervous systems of primitive animals such as insects which have the ability to adapt themselves to a complex environment.

It can be seen that neural networks possess potential for enhancing the capacity of multisensor data fusion, since they have an ability to learn from signal environments and to generalize. A network is said to *generalize* well when the input-output relationship computed by the network is correct (or nearly so) for input-output pattern (test data) never used in creating or training the network. In a linear prediction network, the generalization is nothing but an interpolation. Data contaminated by unknown noise can be used to train the neural network, and the signal signature and noise distribution feature are stored in the connections. Once neural networks are taught well, the association between input patterns (test data) and output patterns is implemented in real time. This capacity makes them play an important role in identification and classification. This ideal will be highlighted in the design of target classification networks.

A neural network is a massively parallel distributed processor. It resembles the brain in two respects: (i) knowledge is acquired by the network through a learning process. (ii) interneuron connection strengths known as synaptic weights are used to store the knowledge. The paradigmatic strength of neural networks for potential applications. which require solving untractable computational problems or adaptive modeling. arises from their spontaneously emergent ability to achieve functional synthesis. Therefore. neural networks may be considered a potential approach to implementing data fusion. The main features of neural networks are. as mentioned above. their ability to learn from the natural environment . and the fact that they are inherently non-linear. Techniques common to neural networks are supervised learning. self-organized learning. and combinatorial optimization. The objectives of our research consist of investigating and developing ANN algorithms for data fusion. carrying out performance analyses and developing software for implementing data fusion. The work has focused on level 1 data fusion. In particular. data association. target position fusion and target classification are studied.

In Chapter 2. two basic models of a neuron. the McCulloch Pitts model and the Fukushima model. are introduced. The neural network is composed of different neurons interconnected by synaptic weights. According to the interconnections and network characteristics. the classification of neural networks is discussed.

In Chapter 3. three optimization networks of Hopfield network, Boltzmann machine and mean-field-theory machine and their relationship are explored.

In Chapter 4 is presented a new energy function for a mean-field-theory machine which can be used for measurement data association. A convergence theorem is put forward. The critical annealing temperature related to the new energy function is obtained. By computer simulations, the performance of neural data association is studied and is compared with that of a conventional method.

In Chapter 5, we discuss multiple dimensional neural data association, which has a powerful association ability.

In Chapter 6, Multiple target classification by the use of multilayer perceptrons is studied, a neural target classifier is designed.

In Chapter 7, we present a novel data fusion approach–feature mapping data fusion (FMDF), the fusion accuracy of which can reach the Cramer-Rao lower bound. A comparison is made between maximum likelihood fusion and feature mapping data fusion.

In Chapter 8, we conclude by giving the highlights of the conclusions which have been derived from the results obtained herein.

Chapter 2

Basic Models of Artificial Neurons and Networks

2.1 Basic Models of Artificial Neurons

Artificial neurons, also called neuron cells, processing elements or simply nodes, attempt to simulate the structure and function of biological neurons. However, artificial neuron models are not exactly constrained by real neurons and are based only loosely on biology. This stems from the following facts:

1. We do not completely understand the behaviour of real nervous systems because of their complexity.
2. Only part of the behaviour of real neurons is essential to their information processing capacity and part of the behaviour builds up irrelevant side effects.
3. From a technical implementation point of view it will probably be impossible and also inefficient to simulate the full behaviour of real neurons.

4. Artificial neural networks are designed in order to realize very specific computational problems. and their architectures and features depend on the problem to be solved.

Below we will describe the basic models used for describing artificial neurons. The use of a particular model depends on the applications in which the algorithms used are close to neural information processing principles.

2.1.1 Basic Neuron Model–McCulloch-Pitts Model

The basic artificial neuron can be modelled as a multi-input nonlinear device with weighted interconnections w_{ji}, also called synaptic weights, as shown in Fig.2.1. The cell body is represented by a nonlinear limiting or threshold function $\varphi(u_j)$. The simplest model of an artificial neuron sums the n weighted inputs and passes the result through a nonlinearity according to the equation

$$y_j = \varphi\left[\sum_{i=1}^{n} w_{ji}x_i + I_i\right] . \tag{2.1}$$

where φ is a limiting or threshold function. called an activation function. I_i is the external threshold. also called an offset or bias. w_{ji} are the synaptic weights. x_i $(i = 1.\dots.n)$ are the inputs. n is the number of inputs and y_j represents the output. Note that a threshold value I_i may be introduced by employing an additional input x_0 equal to $+1$ and the corresponding weight w_{j0} equal to the threshold value. So we can write Eq.2.1 as

$$y_j = \varphi\left[\sum_{i=0}^{n} w_{ji}x_i\right] , \tag{2.2}$$

where $w_{j0} = I_j$, $x_0 = 1$. The basic artificial neuron is characterized by its nonlinearity and the threshold I_j. For example, the early McCulloch-Pitts model of the neuron used only the binary (hard-limiting) function [21, 22]. In this model a weighted sum of all inputs is compared with a threshold I_j. If this sum exceeds the threshold, the neuron output is set to the "high value" or logic 1, otherwise to the "low value" or logic 0.

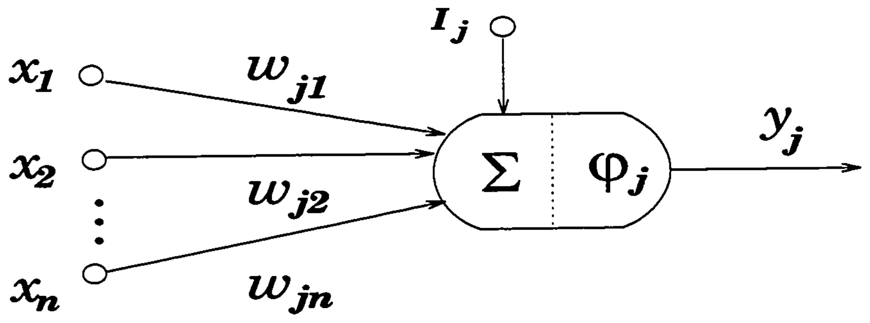


Figure 2.1: Neuron model.

Generally, the threshold function may be replaced by a more general nonlinear function and consequently the output of the neuron y_j can either assume a value of a discrete set (e.g.$\{-1, 1\}$) or vary continuously (usually between -1 and 1). The activation level or the state of the neuron is measured by the output signal y_j. In the basic neural model the output signal is usually determined by a monotonically increasing sigmoid function of a weighted sum of the input signals. Such a sigmoid function can mathematically be described as

$$y_j = \tanh \gamma u_j = \frac{1 - e^{-2\gamma u_j}}{1 + e^{-2\gamma u_j}}, \tag{2.3}$$

for a symmetrical (bipolar) representation or

$$y_j = \frac{1}{1+e^{-\gamma u_j}}, \tag{2.4}$$

for an unsymmetrical unipolar representation with $u_j = \sum_{i=0}^{n} w_{ji}x_i$ where γ is a positive constant or variable which controls the slope of the sigmoidal function. In comparison with the hard limiter, the sigmoid activation function is usually more convenient for analog hardware implementation; moreover, it is more realistic for biological neurons.

A model of the above sigmoid function can be built using traditional electronic circuit components as shown in Fig.2.2. In this very simple circuit a voltage amplifier simulates the cell body, and the variable resistors model the synaptic weights. The sigmoid activation function is naturally provided by the saturating characteristic of the amplifier. The input voltage signals x_i supply current into the wire in proportion to the sum of the products of the input voltages and the appropriate conductances. By applying Kirchhoff's Current Law (KCL) at the input node of the amplifier we obtain the expression

$$y_j = \varphi(u_j) = \varphi\left[\frac{\sum_{i=0}^{n} G_{ji}x_i}{\sum_{i=0}^{n} G_{ji}}\right], \tag{2.5}$$

where $x_i's$ are the input voltages, u_j denotes the input voltage of the jth amplifier, y_j means the output voltage of the jth neuron, $\varphi(\cdot)$ is the sigmoid activation function of the amplifier, $G_{ji} = R_{ji}^{-1}$ is the conductance of the resistor connecting the amplifier i with the amplifier j and I_{ji} is the current flowing through the resistor R_{ji}.

The above equation can be written in the compact form

$$y_j = \varphi\left[\sum_{i=0}^{n} w_{ji}x_i\right], \tag{2.6}$$

where $w_{ji} = G_{ji}/\sum_{i=0}^{n} G_{ji}$.

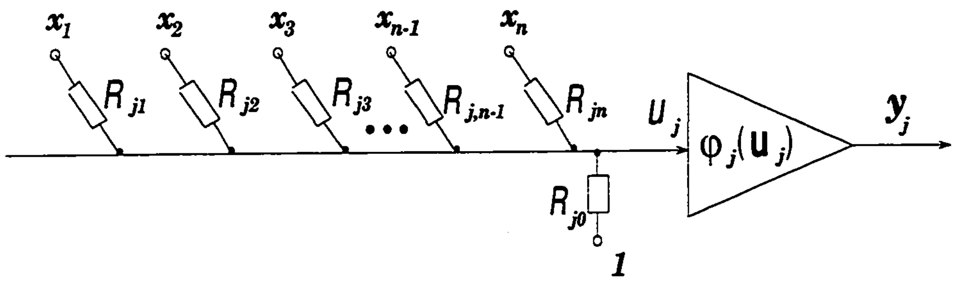


Figure 2.2: Electronic analog model of the basic neuron.

2.1.2 Fukushima Model of the Neuron

The synaptic weights can generally be positive, zero or negative. They are called exciting if they are positive or inhibitory if they are negative. In the model of an artificial neuron suggested by Fukushima all synaptic weights and all input and output signals are nonnegative, i.e. they can take zero or any positive value [23]. In this model the inputs and corresponding synaptic weights are separated into two groups: (i) exciting and (ii) inhibitory. The excitatory part which is the weighted sum of all the excitatory inputs, is suppressed by the inhibitory part, which is the weighted sum of all the inhibitory inputs in a shunting manner. The output of the neuron can be described by the expression

$$y_j = \varphi\left[\frac{1+\sum_{i=1}^{n} a_{ji}x_i}{1+\sum_{i=1}^{m} b_{ji}v_i} - 1\right], \tag{2.7}$$

where

$$\varphi(u_j) = \begin{cases} u_j & \text{if} \quad u_j \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

In this expression the $a'_{ji}s$ mean the excitatory synaptic weights and the $b'_{ji}s$ are the inhibitory synapses. The x'_is and v'_is are excitatory inputs and inhibitory inputs, respectively.

A series of neuron models can be derived from the above neuron structures according to algorithms or learning rules. Usually, the specific neuron structure and activation function are chosen according to the problems to be solved.

2.2 Artificial Neural Network Models

A wide range of models of artificial neural networks (ANNs) has been developed for a variety of purposes; they differ in structure, implementation and principle of operation, but share common features. Generally speaking, artificial neural networks are computing systems made up of a number of simple highly interconnected signal or information processing units (artificial neurons) with the following features:

1. Processing of information and memory are distributed among the whole structure. and therefore it is difficult to separate the hardware and software in the structure. In fact, neural networks are trained, rather than programmed to perform the given task.

2. Artificial neurons are highly interconnected in such a way that the state of one neuron affects the potential of the large number of neurons to which it is connected according to the weights of connection.

3. The connection weights are usually adaptive. Since adaptation can take place everywhere in the structure of the network, we speak of distributed memory.

4. The processing units contain typically nonlinear activation functions, i.e. the new state of the neuron is a nonlinear function of the signals created by the firing activity of the other neurons.

5. Although the networks often use imprecise elements, they are characterized by high robustness to noisy input data and element failure through the use of a highly redundant distributed structure. In other words, neural networks exhibit notable robustness since their functionality is not affected by parameter variations covering a wide range.

There are many different ways to connect artificial neurons to large networks. These different patterns of interconnections between the neurons are called architectures or circuit structures. Such large networks may be able to perform complex tasks which would be impossible for individual neurons. The architectures of artificial neural networks can roughly be divided into three large categories:

- feedforward (multilayer) networks,
- recurrent networks,

- cellular networks.

In feedforward neural networks artificial neurons are arranged in a feedforward manner (usually in the form of layers), i.e., each neuron may receive an input from the external environment or from other neurons, but no feedback is formed. A feedforward network computes an output pattern in response to some input pattern. Once trained the output response to a given input pattern will be the same regardless of any previous network activity. This means that the feedforward neural network does not exhibit any real dynamics, and there are no stability problems in such networks. For feedforward networks the dynamics are often simplified to a single instantaneous nonlinear mapping.

On the other hand, for recurrent (feedback) neural networks the dynamics are no longer trivial, since they consist of processing units with dynamic building blocks (e.g. integrators or unit delays) and they operate in feedback mode. The dynamic properties of such networks are described by a system of nonlinear ordinary differential or difference equations. A feedforward network is represented by static nonlinear maps, while feedback neural networks are represented by nonlinear dynamic systems.

Cellular neural networks consist of special artificial neurons, called cells, which have regular spacing and only communicate directly with their nearest neighbours. Adjacent cells can interact with one another by means of mutual lateral interconnections. Cells not connected together can affect each other indirectly because of the propagation of signals during the transient regime. The cells are usually organized in a two-dimensional array of a rectangular, triangular, hexagonal or other regular grid pattern. Due to local connectivity, every cell is excited by its own signals and by signals flowing from its adjacent cells. Due to mutual interactions the processed signals propagate in time within the whole array of cellular networks.

Generally speaking, an artificial neural network is characterized not only by its architecture, but also by the type of neurons used, as well as by the learning (training) procedure and by the form (principle) of operation. Artificial neural networks can operate, in general, as deterministic or stochastic systems. In deterministic ANNs all parameters and signals have a deterministic nature. In stochastic ANNs signals and parameters are changed randomly (from time to time with the same probability) by some random amount. In the next chapter, we will discuss stochastic neural networks and their direct applications to combinatorial optimization issues.

Chapter 3

Optimization Networks

In many branches of science and technology, difficult optimization problems are often encountered that have combinatorial complexity. For such problems, there is a large but finite set of possible solutions from which we want to find one that globally minimizes the cost function. Typically, if the combinatorial optimization problem is of size n, then the possible solutions are of the order e^n or $n!$. Combinatorial optimization problems are divided into classes according to the computational time needed to solve them. The most important and difficult class of combinatorial problems are **NP**-complete (nondeterministic polynomial time complete) problems [24, 25]. Because of the combinatorial nature of these problems the time needed to solve them grows exponentially and therefore when they have reached a large size they become intractable [25]. For **NP**-complete problems no algorithm is known which provides an exact solution to the problem in a computational time which is a polynomial in the size of the problem. In the last 30 years some heuristic algorithms which provide sub-optimal solutions (in a time that is proportional to a polynomial in the size of problem) have been developed [25, 26]. Unfortunately, known heuristic algorithms for **NP**-complete optimization problems are problem-specific and their implementations on fast parallel computers appear to be very difficult. Recently, a promising new approach has arisen to solve such problems efficiently and in almost real-time by applying neural networks. In the neural network based techniques for combinatorial optimization, the solution consists of finding the minimal point of the neural network energy function. In this chapter, we first review traditional approaches to combinatorial optimization, then we present Hopfield artificial neural networks, Boltzmann machine and mean-field machine in order to pave the way to build our new energy function for data association, which will be given in the next chapter.

3.1 Traditional Approaches to Combinatorial Optimization

Cutting planning technique is one of the methods of solving combinatorial optimization, originally due to Gomory [27]. From an integer linear programming perspective, the idea behind this method is to successively add extra constraints which do not exclude any feasible integer points from the constraint polytope. After each such "cut" has been added, the LP-relaxation solution is found using the simplex algorithm. At each step, either the LP-relaxation solution is an integer, at which point the problem is solved, or else the LP-relaxation solution is used to generate a new cut. Cutting plan algorithms describe how to generate cuts so that no feasible points are excluded at each cut, and so that the algorithm converges in a finite (exponentially bounded) number of steps.

In addition to cutting plan techniques, there are enumerative methods, based on intelligent enumeration of all feasible solutions. The most common enumerative procedure is *branch and bound.* A good survey of the techniques is given by [28]. At each stage of the branch and bound procedure, the set of possible solutions is partitioned into ever smaller mutually exclusive sets: this is the branch operation. An efficient algorithm is then used to compute a lower bound on the cost of any solution in each set: this is the bound operation. As the sets become smaller, and clearly in the limit the set will contain only a single solution, it becomes possible to identify the best feasible solution in a set: at this point exploration of this set can cease. Exploration of a set can also be halted if the lower bound is inferior to any feasible solution found so far. Eventually, it is possible to stop exploring all the remaining solution sets, for one of the above two reasons. At this point the problem is solved, since the best feasible solution found must be the optimal solution to the problem. The branch-and-bound algorithm provides a way of enumerating all feasible solutions without having to consider each and every one. However, this can still take some time, and it is often necessary to terminate the branch-and-bound algorithm before optimality is reached. In such a situation a lower bound on the optimal solution is available.

Another enumerative technique is *dynamic programming*, which is well covered in [29]. Here it is necessary first to express the problem as a multistage decision process: a process in which a sequence of decisions is made, the available choices being dependent on the previous decisions. Dynamic programming exploits the principle of optimality:

> An optimal sequence of decisions has the property that whatever the initial state and initial decision may be, the remaining decisions must be an optimal sequence of decisions with regard to the state resulting from the first decision.

Essentially, this means that the solutions to subproblems can be used to prune the search for the solutions to larger subproblems, and finally to the problem itself. In practice, enumerative methods have found many more applications than cutting plane methods. A full description of all these approaches can be found in an integer programming text [32].

When exact techniques still take too long, inexact heuristics, tailored to particular problems, often perform extremely well. A good example is *local search.* Starting with some feasible solution, a subroutine is called to search for improved solutions within a small neighborhood of the initial solution. The best improved solution is stored, and the subroutine is called again in an attempt to improve this new solution. When no further iterative improvement is possible, the algorithm halts. The skill in designing local search algorithms lies in identifying suitable small neighborhoods in which to search for improvements. With a good choice of neighborhood, local search can perform well [33]. Similar local search techniques also perform well on graph partitioning problems.

3.2 Hopfield Artificial Neural Networks

3.2.1 Analog Model

The Hopfield network belongs to the class of feedback neural networks in which the dynamics are no longer trivial, but play an important role. The dynamics of such networks are described by a system of nonlinear ordinary differential equations and by an associated computation energy (also called Lyapunov or potential) function which is minimized during the computation process. The basic Hopfield model can be implemented by interconnecting an array of resistors, nonlinear amplifiers with symmetrical outputs and external bias current sources as shown in Fig.3.1. The neural network depicted in Fig.3.1 consists of n fully interconnected artificial neurons. Note that a second inverting output of each amplifier is required to implement negative (inhibitory) connection weights. The required weighted sum is produced by superimposition of the resistor currents at the input of each nonlinear amplifier. More precisely, the mathematical model can be derived from Kirchhoff's Current Law (KCL) applied to every input node of the amplifier as

$$C_j \frac{du_j}{dt} = \sum_{i=1}^{n} G_{ji}(x_i - u_j) + I_j^c - \frac{u_j}{R_{j0}}, \tag{3.1}$$

which can be rewritten as

$$C_j \frac{du_j}{dt} = -\frac{u_j}{R_j} + \sum_{i=1}^{n} G_{ji} x_i + I_j^c, \qquad j = 1, \ldots, n,$$

where $x_j = \varphi_j(u_j)$, $C_j > 0$ is the capacitance, u_j and x_j are the internal and output voltages of the neuron j, respectively, I_j^c is an external current applied to the input

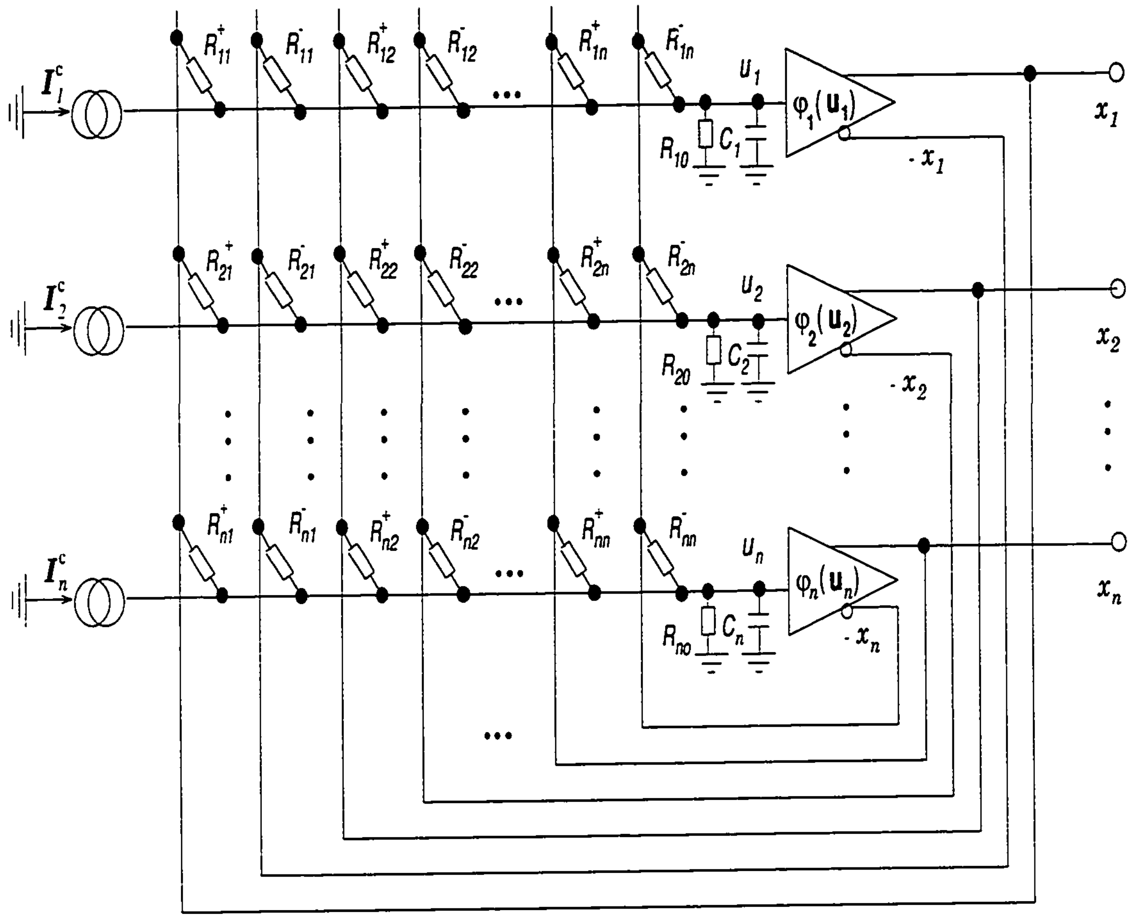


Figure 3.1: Hopfield model of neural networks.

of neuron j. G_{ji} is the conductance representing the synaptic weight from neuron i to neuron j defined as

$$G_{ji} = \frac{1}{R_{ji}^+} - \frac{1}{R_{ji}^-}, \qquad R_{ji}^{\pm} > 0;$$

$$\frac{1}{R_j} = \frac{1}{R_{j0}} + \sum_{i=1}^{n} \left[\frac{1}{R_{ji}^+} + \frac{1}{R_{ji}^-} \right],$$

and $\varphi_j(u_j)$ is the nonlinear, differentiable, monotonically increasing activation function; typically it is defined as

$$\varphi_j(u_j) = [1 + e^{-\gamma_j u_j}]^{-1} \quad \text{or} \quad \varphi_j(u_j) = \tanh(\gamma_j u_j). \tag{3.2}$$

The above system of differential equations can be written in the more convenient normalized form

$$\tau_j \frac{du_j}{dt} = -\alpha_j u_j + \sum_{i=1}^{n} w_{ji}\varphi_i(u_i) + I_j, \quad j = 1, \dots, n, \tag{3.3}$$

where

$$\tau_j = r_j C_j, \quad \alpha_j = \frac{r_j}{R_j}, \quad w_{ji} = r_j G_{ji}, \quad I_j = r_j I_j^c,$$

r_j is the scaling resistance ($\alpha_j = 1$ in the special case $r_j = R_j$). The system of differential equations can be written in the compact matrix form

$$\tau \frac{d\mathbf{u}}{dt} = -\alpha\mathbf{u} + \mathbf{W}\varphi(\mathbf{u}) + \mathbf{I}, \tag{3.4}$$

where

$$\tau = \operatorname{diag}(r_1C_1, r_2C_2, \dots, r_nC_n),$$

$$\alpha = \operatorname{diag}(\alpha_1, \alpha_2, \dots, \alpha_n),$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix},$$

$$\varphi(\mathbf{u}) = [\varphi_1(u_1), \varphi_2(u_2), \dots, \varphi_n(u_n)]^T,$$

$$\mathbf{I} = [I_1, I_2, ..., I_n]^T.$$

In the original Hopfield model the matrix $\mathbf{W}$ of the synaptic weights is symmetrical with diagonal elements equal to zero [34, 35].

The dynamics of the network are determined by the values of the capacitances C_j and the resistances R_{ji} used. Note that the dynamics of the nonlinear amplifiers are assumed negligible. The shape of the nonlinear characteristic of each amplifier is determined by the gain $\gamma_i > 0$ which adjusts the slope or growth rate of the function. Generally, the parameter γ_i is not fixed, but can be changed during the computation process. The connection weights which are determined by the conductances G_{ji} of the corresponding resistors R_{ji}^{+} or R_{ji}^{-} are also generally not fixed, but they can be variable, i.e. electronically reprogrammed for specified tasks.

The set of equilibrium points of the Hopfield neural network is determined from the system of differential equations (3.3) by taking $du_j/dt = 0$, i.e. from the set of nonlinear equations

$$-\alpha_j u_j + \sum_{i=1}^{n} w_{ji}\varphi_i(u_i) + I_j = 0, \quad j = 1, ..., n. \tag{3.5}$$

Hopfield has shown that a sufficient condition for the stability of the network is that the synaptic weights are symmetric i.e. $w_{ji} = w_{ij}$ with $w_{ii} = 0$.

It has been shown that the stable states of a network are the local minima of the computational energy function which is constructed as the first integral of the dynamic differential equations (3.3) [34, 35, 36, 37]

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ji}x_i x_j - \sum_{j=1}^{n} x_j I_j + \sum_{j=1}^{n}\frac{\alpha_j}{\gamma_j}\int_0^{x_j}\varphi_j^{-1}(x)dx, \tag{3.6}$$

which can be written in the more compact matrix form

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x} - \mathbf{x}^T\mathbf{I} + \sum_{j=1}^{n}\frac{\alpha_j}{\gamma_j}\int_0^{x_j}\varphi_j^{-1}(x)dx,$$

where $\mathbf{x} = [x_1, x_2, ..., x_n]^T$, $\mathbf{I} = [I_1, I_2, ..., I_n]^T$, $\alpha_j = r_j/R_j$, and $\varphi_j^{-1}(x_j)$ is the inverse function of the activation function $x_j = \varphi_j(u_j)$. The first two terms of the so formulated energy function correspond to a quadratic cost function which, when minimized, will yield the desired solution to the problem of interest. The last term of the energy function is usually not used in the design procedure of neural networks, and its value depends on the specific shape of the nonlinear activation function φ_j. For high positive gain γ_j, as the slope of $\varphi_j(u_j)$ approaches infinity at $u_j = 0$, the activation function will approach the signum function given by

$$\varphi_j(u_j) = \text{sign}(u_j) = \begin{cases} -1 & \text{if } u_j < 0, \\ +1 & \text{if } u_j > 0. \end{cases} \tag{3.7}$$

In practice, the integral of $\varphi_j^{-1}(x)$ can be neglected since its value is very small. If γ_j $(j = 1, ..., n)$ takes on very large positive values, the nonlinear amplifiers used in the network Fig.3.1 can be viewed as hard-limiter elements or switches. In this case the computational energy function simplifies to the form

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x} - \mathbf{x}^T\mathbf{I}. \tag{3.8}$$

The stable points correspond exactly to the local minima of the computational energy function associated with the network. The operation of the Hopfield neural network can be considered as a minimization computation process of the energy function.

An important property of the Hopfield neural network is its guaranteed convergence to stable states (often interpreted as stored memory) under the sufficient condition that the matrix W is symmetrical.

Lyapunov's stability theory requires the energy function $E(\mathbf{x})$ to be monotonically decreasing in time. Consider the time derivative of the energy function given by Eq.3.6

$$\begin{aligned}
\frac{dE}{dt} &= \sum_{j=1}^{n} \frac{\partial E}{\partial x_j} \frac{dx_j}{dt} = -\sum_{j=1}^{n} \frac{dx_j}{dt} \left[\sum_{i=1}^{n} w_{ji} x_i + I_j - \alpha_j u_j \right] \\
&= -\sum_{j=1}^{n} \tau_j \frac{dx_j}{dt} \frac{du_j}{dt} = -\sum_{j=1}^{n} \tau_j \frac{du_j}{dx_j} \left[\frac{dx_j}{dt} \right]^2 \\
&= -\sum_{j=1}^{n} \tau_j [\varphi_j^{-1}(x_j)]' \left[\frac{dx_j}{dt} \right]^2 . \qquad (3.9)
\end{aligned}$$

Since the time constant $\tau_j = r_j C_j$ is positive for all j and the nonlinear inverse function $\varphi_j^{-1}(x_j)$ is monotonically increased, we can write

$$\frac{dE}{dt} \leq 0. \qquad (3.10)$$

$dE/dt = 0$ implies $dx_j/dt = 0$ for all j. Combining this with the fact that the energy $E(\mathbf{x})$ is bounded, we conclude that the network converges to a stable state which is a local minimum of $E(\mathbf{x})$.

The Hopfield analog model is one of the most popular models of artificial neural networks today and has already found many applications. Moreover, it can easily be implemented by VLSI electronic circuits. However, this model suffers in some applications from several deficiencies, i.e.

1. Whenever the value of a resistor R_{ji}^{+} or R_{ji}^{-} is altered to adjust the corresponding value of the synaptic weight w_{ji}, the total resistance R_j, which determines the coefficient α_j, changes. Thus it is impossible to independently adjust the parameters of the network.

2. The internal potential u_j may assume very large values and, therefore, scaling may pose a problem in practical implementations.

3. The set of equilibrium points is determined by a set of nonlinear equations which are rather difficult to solve and, therefore, it is hard to check the performance of the system.

4. The computational energy function may pose many unnecessary (spurious) local minima which are difficult to avoid.

Mainly for these reasons, many modifications of the Hopfield model have been proposed [38]. The modified models can directly be obtained from the Hopfield model by replacing all lossy integrators and nonlinear amplifiers by ideal (lossless) integrators with saturation. With the modifications, the network has some flexible performance, however, it may still pose many spurious local minima.

3.2.2 Discrete Model

Let us now focus our attention on the dynamical discrete-time Hopfield models. Discrete-time models can be derived from continuous-time models (3.3) by considering the system of nonlinear equations (for $du_j/dt = 0$)

$$-\alpha_j u_j + \sum_{i=1}^{n} w_{ji} x_i + I_j = 0, \quad j = 1, .., n, \tag{3.11}$$

where $x_i = \varphi_i(u_i)$. For iteratively solving the above set of nonlinear equations we can apply the relaxation method to get

$$u_j^{(k+1)} = \frac{1}{\alpha_j} \left[\sum_{i=1}^{n} w_{ji} x_i^k + I_j \right], \tag{3.12}$$

where $u_j^k = u_j(k\tau)$ with the sampling period τ. Assuming for simplicity and without loss of generality that $\alpha_j = 1$ for all j and taking account that $x_j = \varphi_j(u_j)$ we can write

$$x_j^{(k+1)} = \varphi_j \left[\sum_{i=1}^{n} w_{ji} x_i^k + I_j \right], \quad k = 0, 1, 2, ..., \tag{3.13}$$

where $x_j^k = x_j(k\tau)$. In the digital implementation of the discrete-time model a hard-limiting quantizer (signum function) is usually used as the nonlinear activation function since it is more easily realized and computed than any other differentiable sigmoid nonlinearity. So Eq.3.13 can be written as

$$x_j^{(k+1)} = \text{sign} \left[\sum_{i=1}^{n} w_{ji} x_i^k + I_j \right] = \begin{cases} 1, & \text{if } u_j^{(k+1)} \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad k = 0, 1, ... \tag{3.14}$$

For this discrete-time model the associated computational energy function can be defined as

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^T \mathbf{W} \mathbf{x} - \mathbf{x}^T \mathbf{I}. \tag{3.15}$$

where $\mathbf{x} = [x_1, x_2, ..., x_n]^T$, $\mathbf{I} = [I_1, I_2, ..., I_n]^T$, $\mathbf{W} = [w_{ij}]_{n\times n}$ with $w_{ji} = w_{ij}$. Every neuron can take on the values $+1$ or -1, depending on which of the two possible states it is in. The next state of the whole network is computed from the current state by performing the evaluation according to Eq.3.14 on a subset of the neurons to be denoted by $\{N\}$ [39]. The number of neurons of this set can vary from 1 to n. The discrete-time neural network can assume different models during its operation (depending on the number of neurons which can change their state in each time interval). If the computation is performed with all neurons in the same time, we say that the network operates in a *fully parallel mode.* If the computation is performed on a single neuron in any time interval, then we say that the neural network operates in a *serial mode.* All the other cases with $1 < N < n$ (where N denotes the number of neurons in the set $\{N\}$) are simply called parallel modes of operation. The set $\{N\}$ can be chosen randomly or according to some deterministic rule. The convergence property and stability of a discrete-time neural network depend on the model of operation and the structure of the matrix W of synaptic weights.

Let us consider a discrete-time neural network operating in the serial mode. Suppose that x_j is changed to $x_j^{(k+1)} = x_j^k + \triangle x_j$ at some arbitrary time $t = (k+1)\tau$. The resulting change in the energy function given by Eq.3.15 can be expressed as

$$\begin{aligned}\triangle E(\mathbf{x}) &= E(\mathbf{x}^{(k+1)}) - E(\mathbf{x}^k)\\ &= -\triangle x_j \frac{1}{2}\left[\sum_{i=1}^{n} w_{ji}x_i^k + \sum_{l=1}^{n} w_{lj}x_l^k\right] - \frac{1}{2}w_{jj}(\triangle x_j)^2 - \triangle x_j I_j. \qquad (3.16)\end{aligned}$$

Assuming that the interconnection matrix W is symmetric, we can write

$$\triangle E(\mathbf{x}) = - \triangle x_j \left[\sum_{i=1}^{n} w_{ji} x_i^k + I_j \right] - \frac{1}{2} (\triangle x_j)^2 w_{jj}. \tag{3.17}$$

Note that the term in brackets is exactly the argument of the signum function in Eq.3.14 and, therefore, the signs of $\triangle x_j$ and the sign of the term in brackets are the same (or $\triangle x_j = 0$). From this consideration, it can be seen that the energy function is nonincreasing if the interconnection matrix W is symmetric with nonnegative elements in the main diagonal ($w_{jj} \geq 0$). Since the energy function is bounded from below, the network will always converge to a stable state which corresponds to a local minimum in the energy function. Up to now, we can conclude that both analog and discrete-time models of the Hopfield network converge to a local minimum. In physical applications, we will need to find the global minimum. It will be shown in the next section that this can be carried out using the Boltzmann machine.

3.3 Boltzmann Machine and Simulated Annealing

A large class of combinatorial optimization problems can be solved by the Hopfield model of neural networks described by the computational energy function

$$E(\mathbf{x}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j - \sum_{i=1}^{n} x_i I_i = -\frac{1}{2} \mathbf{x^T W x} - \mathbf{x^T I}, \tag{3.18}$$

where the vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ represents the state of the neural network, the matrix $\mathbf{W} = [w_{ij}]$ is a symmetric matrix which represents the synaptic weights between the neurons and vector $\mathbf{I} = [I_1, I_2, ..., I_n]^T$ contains the input bias signals. Let us consider the discontinuous Hopfield model of a neural network which contains n binary neurons. Each neuron can take on only two states, i.e. ON (+1) and OFF (-1); this means $x_i = +1$ or $x_i = -1$. With symmetric connection weights ($w_{ij} = w_{ji}$) the local minima of the energy function are reached at the stable states. A local minimum can be reached when the network is iterated from an initial state by updating each neuron asynchronously in accordance with the updating rule

$$x_i^{(k+1)} = sign\left[\sum_{j=1}^{n} w_{ij}x_j^k + I_i\right], \qquad i = 1, ..., n. \tag{3.19}$$

We recall here that asynchronous iterations are performed in such a way that at each discrete-time point $t_{k+1} = (k+1)\tau$ only one new value $x_i(t_{k+1}) = x_i^{(k+1)}$ is generated leaving the other x_j ($j \neq i$) to be computed during a future iteration. In contrast. for synchronous iterations at time t_{k+1} a new value $x_i^{(k+1)}$ is generated for every x_i using the values x_i^k which were computed at the last iterations. A common pathology of the synchronous mode of operation of the discrete-time mode is a limit cycle behaviour (i.e. the solution oscillates between two states) which often occurs in nonlinear discrete-time systems. The important advantage of the asynchronous mode of operation is the fact that the neural network always converges to a stable state which corresponds to a local minimum of the energy function.

The energy function $E(\mathbf{x})$ may contain many local minima, so that it may be very difficult to find a good solution using the algorithm given by Eq.3.19 which is only guaranteed to converge to the nearest local minimum. In order to escape from a bad local minimum one may be forced to use more sophisticated optimization strategies than the gradient descent. There exist different stochastic procedures for performing the hill-climbing necessary to avoid getting stuck in a local minimum. A frequently exercised and promising approach is the Boltzmann machine [40]. The Boltzmann machine is a kind of stochastic feedback neural network consisting of binary neurons connected mutually by symmetric weights [40, 41, 47, 48]. In fact the Boltzmann machine is an energy minimization network consisting of statistical neurons which appear probabilistically in one of two states ON or OFF(e.g. $+1$, -1 or 1, 0). The algorithm used by the Boltzmann machine to locate the energy function minima follows the simulated annealing approach [40, 42, 43, 44]. Simulated annealing is a stochastic strategy for searching the state of neurons corresponding to the global minimum of the energy function (3.18). This strategy has an analogy to the physical behaviour of annealing of a molten solid [42, 45]. At a high temperature all particles (atoms) of a metal lose the solid phase so that the positions themselves are random according to statistical mechanics (i.e. at a high temperature the particles are in violent random motion). As with all physical systems the particles of the molten metal tend toward the minimum energy state, but a high thermal energy prevents this. The minimum energy state usually means a highly ordered state such as a defect-free crystal lattice. In order to achieve the defect-free crystal the metal is annealed, i.e. at first it is heated to an appropriate temperature above the melting point and then cooled slowly until the metal freezes into a "good" crystal. The slow cooling is usually necessary to prevent dislocations and other crystal lattice disruptions. The metal having a random thermal energy must also be gradually and carefully cooled down in order to reach the defect-free crystal state corresponding to the global minimum of the thermal energy.

The Boltzmann machine introduces artificial thermal noise, whose amplitude is gradually decreased with time. This noise allows occasional hill-climbing interspersed with descents. Strictly speaking, the fluctuations of the energy function $E(\mathbf{x})$ are allowed to be a Boltzmann probabilistic distribution (hence the name of the network)

$$P(\mathbf{x}) = \frac{\exp\left[-\frac{E(\mathbf{x})}{T}\right]}{\sum_{\mathbf{x}} \exp\left[-\frac{E(\mathbf{x})}{T}\right]}, \tag{3.20}$$

where the sum runs over all possible configurations of the states, and T is a controlling parameter called the computational temperature. In physical systems the temperature T has physical meaning; in the Boltzmann machine the temperature is simply a parameter which controls the magnitude of fluctuations of the energy function $E(\mathbf{x})$. The idea is to apply uniform random perturbations to the output states of the neurons and then determine the resulting change $\triangle E$ in the energy. If the energy is reduced the new configuration is accepted. However, if the energy is increased the new configuration may also be accepted but with a probability proportional to $\exp(-\triangle E/T)$. In other words, we must select a random number N_r between zero and one using a uniform density function. If $N_r < \exp(-\triangle E/T)$, then the new state is accepted, otherwise it is rejected. At a high temperature the probability of the energy function moving uphill is large. However, at a low temperature the probability is low, i.e. as the temperature decreases fewer uphill moves are allowed. The simulated annealing allows moves uphill in a controlled fashion, so there is no danger of jumping out of a local minimum and falling into a worse one.

For the energy function (3.18) the probability of a particular neuron being in the ON(+) or OFF(-1) state is given by [47, 48]

$$P(x_i = \pm 1) = \frac{\exp(\pm u_i/T)}{\exp(u_i/T) + \exp(-u_i/T)}, \qquad (3.21)$$

$$\text{where } \; u_i = \sum_{j=1}^{n} w_{ij} x_j + I_i, \qquad x_i \in \{-1, 1\} \;\; \forall i.$$

In practice the probabilistic acceptance or rejection of the state is achieved by adding to each neuron a separate "thermal" noise component N_i. So the output state of any neuron can be computed as

$$x_i^{(k+1)} = \tanh\left[\gamma \left[\sum_{j=0}^{n} w_{ij} x_j^k + N_i\right]\right], \;\text{ where } \quad \gamma \gg 1,$$

where x_0^k is the input bias signal. The gain γ should be high enough so that the sigmoid activation function closely approximates the signum function. Each neuron should be fed by an additive zero-mean independent noise source in such a way that its state will be unaffected by the noise applied to the other neurons. The noise must be slowly reduced in time in order to perform a process of simulated annealing. The efficiency of the simulated annealing approach crucially depends on the choice of the cooling schedule for the control parameter T. Similar to the diffusion optimization algorithm in the standard annealing schedule the temperature T is inversely proportional to a logarithmic function of time. Such a temperature cooling schedule is rather slow, often too slow to be practical. Generally speaking, if the cooling schedule is too slow, a satisfactory solution might never be reached, and if it is too fast, a premature convergence to a local minimum might occur. The problem of accelerating the simulated annealing algorithm is the subject of current active research. The simulated annealing algorithm can be performed as follows:

1. Get an initial system configuration; begin with an arbitrary state $\mathbf{x}(n)$.

2. Define a parameter T which represents a computational temperature, start with T at a large value.

3. Make a small random change in the state.

4. Evaluate the resulting change in the energy function $E(\mathbf{x})$.

5. If the energy function is reduced, retain the new state, otherwise accept the transition to the new state with the probability $P = \exp(-\triangle E/T)$. For this purpose select a random number N_r from a uniform distribution between zero and one. If $P(\triangle E)$ is greater than N_r retain the new state, otherwise return to the previous state.

6. Repeat steps 3 through 5 until the system reaches an equilibrium, i.e. until the number of accepted transitions becomes insignificant.

7. Update the temperature T according to an annealing schedule and repeat steps 3 through 6. In practice, the factor used for decreasing the temperature from one step to another is chosen as 0.85 to 0.96.

The algorithm stops when the temperature is small enough to consider the system to have reached a state near ground level (absolute minimum). The main drawback of the simulated annealing algorithm is a very long computation time, since it is necessary to perform a large number of random searches (elementary transformations) at each temperature step to arrive near the equilibrium state. One promising approach for speeding up the convergence in order to arrive at an optimal solution is to apply the mean-field theory [47].

3.4 Mean-Field-Theory Machine and Annealing Algorithm

In [49], Peterson demonstrated that the stochastic simulated annealing process in the Boltzmann machine can be replaced by a set of deterministic equations in the so called mean-field-theory (MFT) approximation. This mean-field-theory learning algorithm typically provides a substantial speed-up over the Boltzmann machine. For feature recognition problems, the mean-field-theory requires a substantially small number of training epochs than BP and provides as good generalization properties as that obtained by BP. Peterson[48] applied the mean-field-theory idea into TSP. This method limits the redundancy of measurements within the neural network, a small population of observations are used and the quality of the solution for TSP is in parity with those obtained by using optimally tuned simulated annealing heuristics. Clearly, the measurements in the stochastic network (Boltzmann machine) are statistically random. Replacing a large amount of random measurements by the mean of the measurements is a shortcut to finding a neuron stable state. The MFT machine is to use a small population of observations to achieve as good a solution as that obtained by the Boltzmann machine which uses a large amount of measurements within the neural network.

The Mean-field-theory machine is a kind of deterministic feedback neural network consisting of nonlinear continuous neurons connected mutually by mean fields (functions of mean values of the binary neuron states). In contrast to simulated annealing, which is a purely stochastic algorithm, the mean-field-theory (MFT) procedure is deterministic [47, 48]. The basic idea is to replace stochastic binary neurons, say with two binary states $x_i = \pm 1$, by analog neurons with continuous outputs v_i constrained between -1 and 1. More precisely, the continuous variable v_i of the *ith* analog neuron will be determined as the average (mean) value of the *ith* binary neuron variable x_i at the temperature T, which can be written mathematically as

$$v_i := < x_i > |_T = P(x_i = +1) - P(x_i = -1) = \tanh\left[\frac{u_i}{T}\right]. \tag{3.22}$$

Assuming that the energy function, called the free energy, of the analog model of the neural network is defined as

$$E(v) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}v_iv_j - \sum_{i=1}^{n} v_iI_i = -\frac{1}{2}v^{\mathbf{T}}\mathbf{W}v - v^{\mathbf{T}}\mathbf{I}, \tag{3.23}$$

the MFT approximation takes the form

$$v_i = \tanh\left[\left[\sum_{j=1}^{n} w_{ij}v_j + I_i\right]/T\right] = \tanh\left[-\frac{1}{T}\frac{\partial E(v)}{\partial v_i}\right], \quad i = 1, \dots, n. \tag{3.24}$$

In this way the complex stochastic process of simulated annealing has been approximated by a system of nonlinear deterministic equations called mean-field annealing (MFA) equations [48]. We denote $\frac{\partial E(v)}{\partial v_i}$ as the mean field, which is the connection between the neurons. The solution of these equations should give the global minimum of the energy function $E(v)$, under the assumption that the time schedule of changing the controlling parameter T is appropriately chosen. However, mathematically, the rigorous proof of the convergence of the mean-field-theory machine is still a void to be filled. During the annealing process, it has been found [41, 47, 48] that above a certain critical temperature T_c, all the variables v_i remain nearly constant, having approximately the same value. At the critical temperature, the network exhibits a behavior analogous to a phase transition, and all the variables v_i start diverging in value. Since the convergence is very slow, if any, for $T > T_c$, the efficiency of the network can be greatly improved by slowing down the annealing process at $T \cong T_c$.

The MFA equations can be solved by various kinds of neural network models. At first, let us consider the continuous Hopfield model (for details see Section 3.2) described by the system of nonlinear differential equations

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_{j=1}^{n} G_{ij} v_j + I_i,$$

$$v_i = \tanh(\gamma_i u_i), \quad i = 1, ..., n, \tag{3.25}$$

where G_{ij} is the conductance representing the synaptic weight from neuron j to neuron i, I_i is the input external current of the neuron i, u_i is the internal potential of the neuron i, v_i is the output voltage of the neuron i, C_i is the capacitance. $R_i = \left[\frac{1}{R_{i0}} + \sum_{j=1}^{n} G_{ij}\right]^{-1}$ is the total input resistance of the ith neuron, γ_i is the slope of the activation function of the ith neuron. Note that the set of equilibrium points can be determined from the above system of differential equations taking $du_i/dt = 0, \forall i$, i.e.

$$u_i = \left[\sum_{j=1}^{n} G_{ij} v_j + I_i\right] R_i, \quad i = 1, ..., n. \tag{3.26}$$

Taking into account Eq.3.25 and assuming that $\gamma_i = 1/(R_i T)$, we can write

$$v_i = \tanh(\frac{u_i}{R_i T}). \tag{3.27}$$

Hence

$$v_i = \tanh\left[\left[\sum_{j=1}^{n} G_{ij} v_j + I_i\right] / T\right]. \tag{3.28}$$

Note that Eq.3.28 are equivalent to the MFA equation (3.24) at an equilibrium under the assumption that

$$w_{ij} = G_{ij}. \tag{3.29}$$

We have also shown that the continuous Hopfield model with sigmoid activation functions directly simulates the MFA equations, provided that the nonlinear differential equations of continuous Hopfield network are converged.

It is worth mentioning here that the Hopfield neural network with discontinuous (hard-limiter) activation functions has a smaller degree of freedom, since it is constrained to changing the states along the edges of an $n-$dimensional hypercube $\Omega = \{-1, 1\}^n$, thereby increasing the probability of being trapped in a local minima. Replacing the signum activation functions by sigmoid functions has the effect of smoothing out some of the local minima. Especially, if the neurons have a low gain, the network can traverse the interior of a hypercube Ω, but then the discrete final states $\{-1, 1\}$ are not guaranteed. Therefore, it is usually necessary to gradually increase the gains of neurons during the optimization process to provide a discrete final state vector. This means that in the hardware implementation of the Hopfield network we should use amplifiers with tuned gains γ_i. Varying the gain of the neurons (amplifiers) is equivalent to changing the temperature T. If the gain of all neurons is very small (which corresponds to a high temperature T) the neural network behaves approximately as a linear system (since all amplifiers operate in the linear region), so each neuron could take any value between -1 and 1 instead of the well-defined binary state (-1 or 1). However, if the neurons have a high gain γ_i (which corresponds to a low temperature) the states of the neurons will be forced to either the ON or OFF state due to the positive feedback of the network.

There are many different possible implementations of the MFA equations. As an alternative to Eqs.3.25 one can use, for example, the system of differential equations [47]

$$\frac{dv_i(t)}{dt} = -\mu_i \left\{ v_i(t) - \tanh\left[\left(\sum_{j=1}^{n} w_{ij} v_j(t) + I_i\right) /T\right]\right\}, \quad i = 1, \ldots, n. \tag{3.30}$$

where $\mu_i > 0$. The above system of equations can be realized by the simple analog neural network, they can also be solved by using iteration formulas, i.e. by a discrete-time model. For example, by applying the Euler rule

$$\frac{dv_i(t)}{dt} \cong \frac{v_i((k+1)\tau) - v_i(k\tau)}{\tau} = \frac{v_i^{(k+1)} - v_i^k}{\tau}, \tag{3.31}$$

we obtain from Eq.3.30 the iterative algorithm

$$\begin{aligned} v_i^{(k+1)} &= v_i^k - \tau\mu_i \left\{ v_i^k - \tanh\left[\left(\sum_{j=1}^{n} w_{ij} v_j^k + I_i\right) /T\right]\right\}, \\ k &= 0, 1, 2, \ldots, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{3.32}$$

In the special case that $\tau\mu_i = 1, \ \forall i$, the above algorithm simplifies to

$$v_i^{(k+1)} = \tanh\left[\left(\sum_{j=1}^{n} w_{ij} v_j^k + I_i\right)/T\right], \quad k = 0, 1, 2, ..., \quad i = 1, 2, ..., n. \qquad (3.33)$$

Note that for a temperature T close to zero Eqs.3.33 closely approximates Eqs.3.19.

In many applications of neural networks the output states are represented by $\{0.1\}$ instead of $\{-1.1\}$. For this representation of the output signals of neurons the MFA equations 3.24 can be modified as

$$v_i = \frac{1}{2}\left\{1 + \tanh\left[\left[\sum_{j=1}^{n} w_{ij} v_j + I_i\right]/T\right]\right\}, \quad i = 1, ..., n. \qquad (3.34)$$

This means that in hardware realizations of these equations, the bipolar sigmoid activation functions (the hyperbolic tangent) must be replaced by the unipolar logistic activation functions.

The important advantage of the MFA equations 3.24 and 3.34 is that they provide a good estimation of the collective nature of the stochastic neural networks independently of their practical implementations, i.e. without appealing to complicated neural network dynamics. In other words, they provide a shortcut to achieve an optimal solution by the use of a small population of observations within the neural network.

In summary, instead of directly simulating the stochastic network (Boltzmann machine) it is possible to estimate its mean behaviour using the mean-field theory which replaces each stochastic binary variable $x_i \in \{-1, 1\}$ by a deterministic continuous variable $v_i \in [-1, 1]$ $(-1 \leq v \leq 1)$ representing the mean value of the stochastic variable. In this way the simulated annealing procedure is replaced by a deterministic dynamic procedure. The mean-field annealing algorithm exhibits better (quicker) convergence while preserving a nearly equal quality of the solution afforded by the simulated annealing approach. In the next chapter, as a further development of the mean-field theory, we create a new energy function of the mean-field-theory machine designed for multiple target data association.

Chapter 4

Neural Data Association

Multiple-target-tracking (MTT) is an essential requirement for surveillance systems employing one or more sensors, together with computer subsystems, to interpret an environment that includes both true targets and false alarms. With the recent proliferation and the increasing sophistication of new technologies, system designers are recognizing that the incorporation of new techniques, such as neural networks, fuzzy logics and others, into MTT will make the surveillance system more powerful [7], [17], [53]. In MTT, the key component is data association or data correlation. As far as data association is concerned, the probabilistic data association (PDA) and all-neighbours joint probabilistic data association (JPDA) methods are currently popular in the literature related to tracking. Both techniques probabilistically "smooth" or "filter" the data within the gate of interest [53]. The predicted track positions are associated with the smoothed points instead of the real plots.

This chapter develops and analyses the neural network for measurement data association, i.e., assigning plots (measurements of target positions) to predicted track positions. This particular design of neural network can achieve functional synthesis. For a simple example of functional synthesis, we will refer to mathematical function approximation. It has found a wider range of applications in the optimization processing area, particularly in multitarget data association in recent years.

The measurement data association (MDA) problem can be viewed as an example of the typical combinatorial optimization like traveling salesman problem (TSP). In TSP, given the positions of a specified number of cities, assumed to lie in a plane, the problem is to find the shortest path that starts and finishes at the same city. The TSP is thus simple to state but hard to solve exactly, in that there is no known method of finding the optimum tour, short of computing the length of every possible tour and then selecting the shortest one. In a pioneering paper among many publications, Hopfield and Tank (1985) demonstrated how an analog network, based on the system of coupled first-order differential equations, can be used to represent a solution of the TSP. Specifically, the synaptic weights of the neural network are determined by distances between the cities visited on the tour, and the optimum solution to the problem is a fixed point of neurodynamical equations. Herein a key point is how to map the combinatorial optimization problem onto the continuous Hopfield network. The network acts to minimize a single energy (Liapunov) function, and yet the typical combinatorial optimization problem requires the minimization of an objective function subject to some hard constraints. If any of these constraints is violated, the solution is considered to be invalid. The early mapping procedures were based on a

Liapunov function constructed in an intuitive manner, usually employing one term for each constraint, as shown by

$$E = E^{opt} + c_1 E_1^c + c_2 E_2^c + \dots . \tag{4.1}$$

The first term, E^{opt}, is the objective function to be minimized; it is determined by the problem at hand. The remaining terms, E_1^c, E_2^c, ..., represent penalty functions whose minimization satisfies the constraints. The c_1, c_2, ..., are constant weights assigned to the respective penalty functions E_1^c, E_2^c, ..., usually by trial and error. Unfortunately, the many terms in the Liapunov function of Eq.4.1 tend to frustrate one another, and the success of the neural network is highly sensitive to the relative values of c_1, c_2, ...

Hopfield organization networks have been applied to the TSP by various researchers, but with varying degrees of success because of recurring instability and local minimization sticking. Usually, there are five constants to be decided arbitrarily [17] [51]. In practice, it is very difficult to choose the five constants to make sure that the optimization will be achieved. On the other hand, the Boltzmann machine is another alternative for solving the TSP. However, the Boltzmann machine's convergence speed is very slow, even though it can achieve an optimal solution of the TSP.

MDA has been studied by several authors [17]. It can be structured in a basic framework very similar to that of the classic TSP. Therefore the untractable problems in the TSP also exist in MDA.

To cope with these problems, this chapter presents the mean-field-theory machine for MDA which is an alternative of the Hopfield network and the Boltzmann machine, and has the advantages of both. Since the neuron arrangement is the same as for the Hopfield network, we call it mean field Hopfield network. In the chapter, we present the new energy functions and its calculation to get the optimal plot/track association, and derive the critical simulated annealing temperature that is very useful for neural computation. The convergent theorem is put forward, which describes the dynamic of the mean field machine. Then we describe the performance of the mean field Hopfield network for MDA with the different accuracies of measurement data, and discuss the applicability of the neural network. The new technique developed here is denoted as mean-field-theory data association (MFTDA).

4.1 A New Energy Function for MDA and Neural Computation

4.1.1 TSP Network and Comments on Its Solution

Hopfield [35] [37] demonstrated that neural networks can solve optimization problems, such as the TSP, which is similar to the association problem. A major contribution on his part is the demonstration that an appropriately constructed network will find local minima of an energy function which is defined as:

$$E = -\frac{1}{2}\sum_{i}^{N}\sum_{j}^{N} T_{ij}v_i v_j - \sum_{i}^{N} v_i I_i, \tag{4.2}$$

where E is the energy, N is the total number of neurons, v_i is the value of the ith neuron, T_{ij} is a connection weight on the link from the jth neuron to the ith neuron, and I_i is the external excitation applied to the ith neuron. As long as the weights T_{ij} and T_{ji} are equal, the network is symmetric, and E above is a Liapunov function characterizing the network's dynamics, the network seeks a stable equilibrium.

Hopfield showed that if an optimization problem can be mapped into this energy equation, a neural network can be designed to solve it. However, because the network seeks local minima of the energy function, it is not certain that the solution obtained with the Hopfield network is globally optimal.

Fig.4.1 is the simple representation for the solution to the TSP. It shows a square grid with each row corresponding to a city and each column to a position in the tour. Every square in the position (X, i) in the grid represents the event "city X is in position i in the tour." If the output of a neuron in the position (X, i) is "1", the city X will be visited in position i.

The neural network energy function used in Hopfield and Tank (1985) [35] in an attempt to generate a minimal length tour through a set of n cities (i.e. the TSP) is

$$\begin{aligned} E \;=\; & \frac{A}{2}\sum_X\sum_i\sum_{j\neq i} v_{Xi}v_{Xj} + \frac{B}{2}\sum_i\sum_X\sum_{Y\neq X} v_{Xi}v_{Yi} \\ & + \frac{C}{2}(n - \sum_X\sum_i v_{Xi})^2 + \\ & \frac{D}{2}\sum_X\sum_{Y\neq X}\sum_i d_{XY}v_{Xi}(v_{Y,i+1} + v_{Y,i-1}), \end{aligned} \tag{4.3}$$

where the $v's$ are continuous variables in the region [0,1] such that $v_{Xi} \approx 1$ indicates

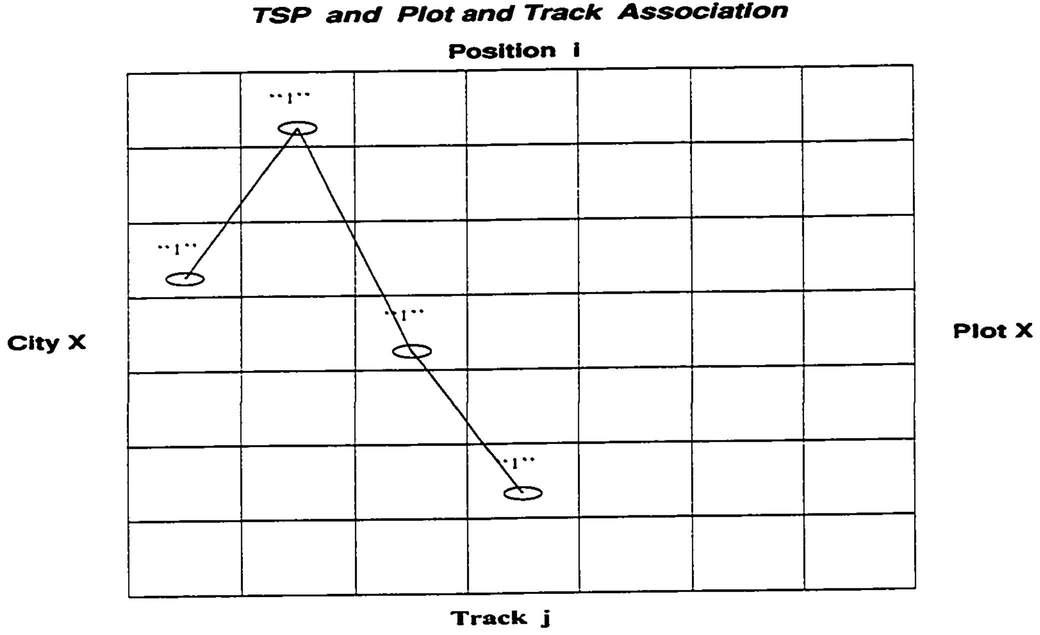


Figure 4.1: Neuron configuration for TSP and MDA.

that city X is the ith city to be visited (otherwise, $v_{Xi} \approx 0$). The first term of Eq.4.3 attempts to insure that any city X does not show up on the final tour in two (or more) positions i and j. The second term biases the final solution such that any two cities X and Y are not assigned to be visited in the same position i on the tour. The third term makes sure that the total tour visits all n cities. The final term sums the distances, d_{XY}, between adjacent cities on the tour so as to find the total tour length. (All tour indexing operations are done modulo-n.) Thus, the first three penalty terms try to maintain a feasible solution while the final term steers the neural network towards those valid tours with the shortest length. Comparing Eq.4.3 to Eq.4.2, it is easy to find neuron interconnection weights $T'_{ij}s$ [17].

The settings for A, B, C, and D determine the priority of each term in the objective function. Setting low values for the parameters A, B, and C emphasizes the length term and usually leads to short, invalid tours. Alternatively, making A, B, and C large stiffens the penalties so much that the network will converge to any feasible solution regardless of its total length. From experimentation, it is known that good values for A, B, C, and D may exist in very narrow, difficult-to-find regions in the parameter space.

Smith [17] and others deployed above energy function to carry out the measurement data association. In Fig.4.1, each row corresponds to a plot and each column to a track. Each square in the grid stands for the event "plot X associates with track j." Using above energy function for MDA, there are five constants to be decided by trial and error. (In addition to A, B, C, D, the n has to be chosen arbitrarily.) It is not feasible to apply this technique to tracking systems.

4.1.2 Derivation of a New Energy Function

A highly attractive feature of the Boltzmann machine is that it can avoid local minima by incorporating a relaxation technique based on simulated annealing into its learning procedure. However, the use of simulated annealing requires excessively large computation times: this fact has hindered experimentation with the Boltzmann machine. Not only does simulated annealing require measurements at a sequence of temperatures that defines the annealing cycle, but also each measurement requires many sweeps of its own. To overcome this major limitation of the Boltzmann machine, we may use a mean-field approximation, according to which the stochastic, binary-state neurons of the Boltzmann machine are replaced by deterministic, analog ones.

The idea of mean-field approximation is well known in statistical physics. In the case of a network with a large number of neurons, the neural states contain vastly more information than we usually require in practice. In fact, to answer the most familiar physical questions about the stochastic behavior of the network, we need only know the average values of neural states. In information processing, using the smaller amount of samples can achieve the same effectiveness as using a large amount of random samples by employing mean fields. In a combinatorial optimization, the constraints are incorporated into an energy function [48]. Therefore, the network needs not learn from outside. In the application of the mean-field-theory to optimization problems, the mean fields act as the links between the neurons. These links will guide the energy function to the lowest value during the evolution of neuron states.

In Eq.4.3, the neurons are fully interconnected by equivalent transition functions [17] [35]. In a mean-field-theory machine, neurons are interconnected by mean fields. Referring to the template of plot/track association in Fig.4.1, and pondering deeply over the Hopfield approach, the third term in Eq.4.3 can be omitted by neural normalization. We can simplify the energy function by means of the mean-field-theory, and avoid the local minimization sticking through simulated temperature annealing. An energy function can be built in such way that its global minimum value corresponds to minimal summation of plot-track distances when the association arrives at an optimal status. It can be considered as a cost function; the cost function takes a minimal value when the plot-track association is in the optimal status. When we deploy the neural normalization [51], the MDA energy function can be written as

$$E = \frac{d_p}{2} \sum_i \sum_X \sum_{Y \neq X} v_{Xi} v_{Yi} + \sum_X \sum_i v_{Xi} \sum_j v_{Xj} d_{Xj}, \tag{4.4}$$

where X, Y denote plot index and i ,j denote track index, v_{Xi} is the neuron output which represents the probability that plot X associates with track i. d_{Xj} is the distance between plot X and track j. and d_p is a penalty constant. The first term maintains feasibility by acting as a repulsive force that discourages two plots from associating with the same track, while the second term accounts for the total distances between plots and tracks when the neuron states arrive at a state of optimal association. These two terms are related through a single parameter, d_p, which is simply set to a value slightly larger than the twice the largest distance between any plot and track. If the value of the constant d_p is set large, the first term in Eq.4.4 is emphasized, otherwise, the second term is emphasized.

Note that no term is provided in Eq.4.4 to penalize a plot which is not assigned or is assigned more than once. This constraint is handled in an explicit manner as follows: Each neuron's function level v_{Xi} is looked upon as the probability that plot X is currently assigned to track i as the plots undergo random thermal perturbations. At a given simulated temperature T, the probability that plot X associates with track i obeys a Boltzmann distribution

$$v_{Xi} \propto e^{-E_{Xi}/T}; \tag{4.5}$$

where E_{xi} is the mean field of a neuron that corresponds to plot X and track i, which can be expressed as

$$E_{Xi} = d_p \sum_{Y \neq X} v_{Yi} + \sum_j v_{Xj}(d_{Xj} + d_{Xi}). \tag{4.6}$$

The neurons are interconnected through the mean field strength. Thus, the tracks with higher mean fields are less likely to be associated than those with lower values which correspond to shorter plot-track distances with fewer violated constraints. In order to be a true probability, the neural function levels are normalized as follows:

$$v_{Xi} = \frac{e^{-E_{Xi}/T}}{\sum_j e^{-E_{Xj}/T}}, \tag{4.7}$$

which guarantees that each plot will be assigned once ($\sum_i v_{Xi} = 1$). The first term in Eq.4.4 guarantees that each track will associate one plot, and the second term in Eq.4.4 stands for the total distances between plots and tracks which will be the shortest when a state of optimization is achieved. Thus, valid and global distance minimization plot-track assignments are guaranteed through the combined action of the new objective function and the explicit normalization procedure when the network arrives at a state of optimization. The network optimization state can be obtained by the use of temperature annealing.

4.1.3 Calculations

The quantity E, given by Eq.4.4, is the objective function that is to be minimized. It depends on a vector argument, $\Upsilon = \{v_{Xi}\}$, and is a super-hyperboloid. The large number of possible combinations of the interacting variables makes searching for the optimal solution very difficult. Here temperature annealing is incorporated into the calculation of the objective function that was originally used by Hopfield and Tank (1985), where the neural gain ($\frac{1}{T}$) was increased as the network evolved. Actually, the temperature annealing is the Boltzmann machine's operational behavior. When T changes, the super-hyperboloid changes, too. This activity overcomes the problem of sticking at a local minimum. The pseudo-code for the mean field Hopfield network calculation is presented as follows: This algorithm begins at a temperature somewhat above the critical point T_c and visits a decreasing series of temperatures as specified by a cooling constant $\alpha < 1$.

```
initialize each neuron
$T \Leftarrow T_c + \triangle T$
while $T > T_c - \triangle T$
          continue until a fixed-point is found
                    select a plot $X$ at random
                    $sum = 0$
                    for $i = 1 : n$
                        $E_{Xi} = d_p \sum_{Y \neq X} v_{Yi} + \sum_j v_{Xj}(d_{Xj} + d_{Xi})$
                        $sum \Leftarrow sum + e^{-E_{Xi}/T}$
                    for $i = 1 : n$
                        $v_{Xi} = e^{-E_{Xi}/T}/sum$
                    $E = \frac{d_p}{2} \sum_i \sum_X \sum_{Y \neq X} v_{Xi} v_{Yi} + \sum_X \sum_i v_{Xi} \sum_j v_{Xj} d_{Xj}$
```

$T \Leftarrow \alpha T$

---

where n is the number of tracks. From the above we can see that this algorithm has the following features:

1. Neurons are fully interconnected by the mean field,
2. The probability that plot X associates track i obeys a Boltzmann distribution.
3. Use of neuron normalization.
4. Use of simulated temperature annealing.
5. Neuron activity levels update non-synchronously.

In Fig.4.2 is given an 8-pair plot-track data association result. In the solution searching process, the cooling constant, $\alpha = 0.9$, was used to make a series of temperature changes. At a low temperature, the neurons come to rest in a stable state, and the energy function achieves the minimal value, $E = 1267$. This value is the total distance of the eight associated plot-track pairs, where global distance minimization is achieved. If, in this example, we deployed the nearest neighbour association, the nearest plot and track would be associated during the first operation. Then in order to complete the eight-pair plot-track association, the plots and tracks which had large separations would have to be linked together. The cost (or risk) would be larger than for the optimal case. In Fig.4.3 is given the nearest neighbour association result.

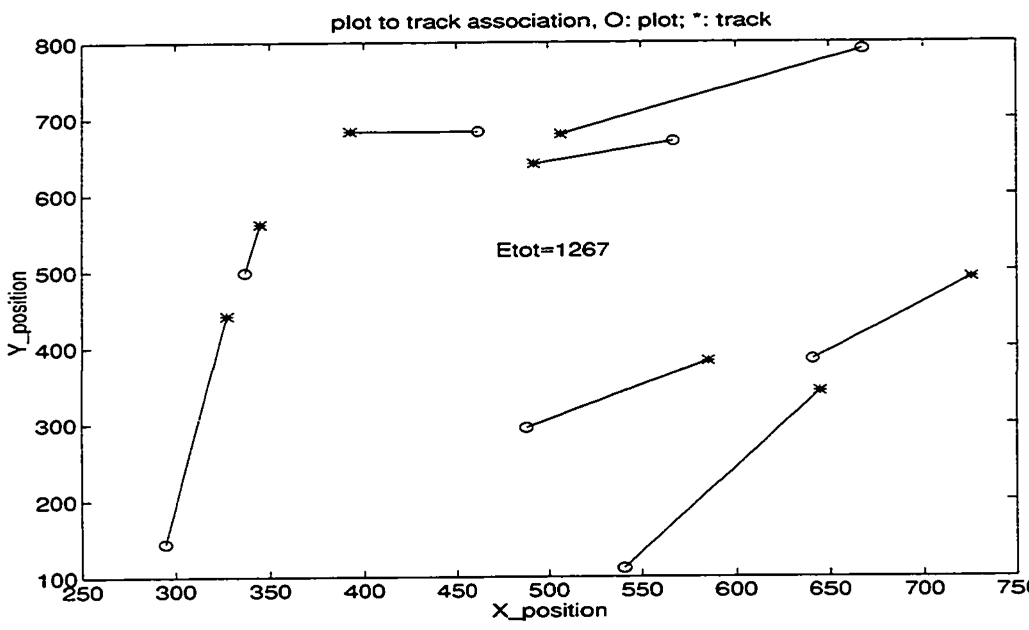


Figure 4.2: Neural data association result for eight plot-track pairs.

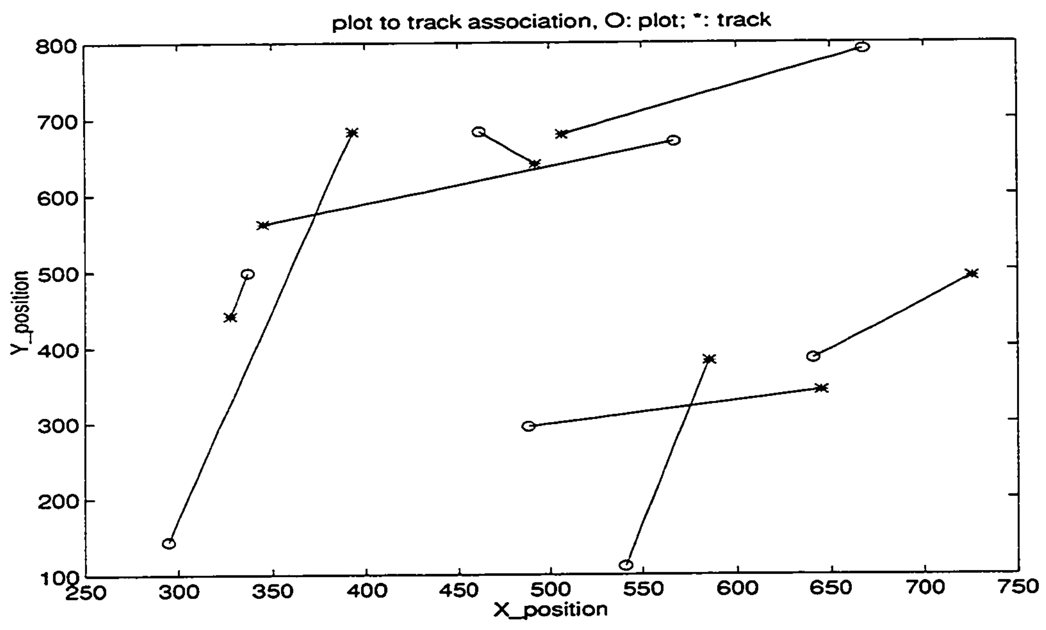


Figure 4.3: Nearest neighbour association result for eight plot-track pairs.

4.2 Critical Annealing Temperature

From Eq.4.5 and Eq.4.7, it can be seen that at high temperatures each plot will be smeared equally across each track position:

$$T \to \infty \Rightarrow v_{Xi} \to \frac{1}{n}, \quad \forall X, i.$$

At lower temperatures, the plots coagulate onto particular tracks, which causes the total distance to be minimized. To gain further understanding of the neural association behavior, let us consider a 15-pair plot/track association example. In this case, the $X - Y$ position coordinate is normalized. Each plot position is generated by a random Gaussian distribution with the corresponding track position being the distribution center. In Fig.4.4 is shown the association result. In Fig.4.5 is presented the neuron activation function signal when the energy function is reduced to its minimum value. Fig.4.6 is the neuron mean field state at the final optimal stage. In Fig.4.7 is presented the variation of the energy function versus annealing temperature. Fig.4.8 shows the variation of the energy function against the number of iteration. Figs.4.4-4.8 describe the network evolving process in detail. In this example, as the temperature varies from a large value to a small one, the value of the energy function drops and finally the neuron mean fields are in their stable state, i.e., the neurons corresponding to optimal association are saturated. At this stage, the network remains at the equilibrium point which is the optimal one.

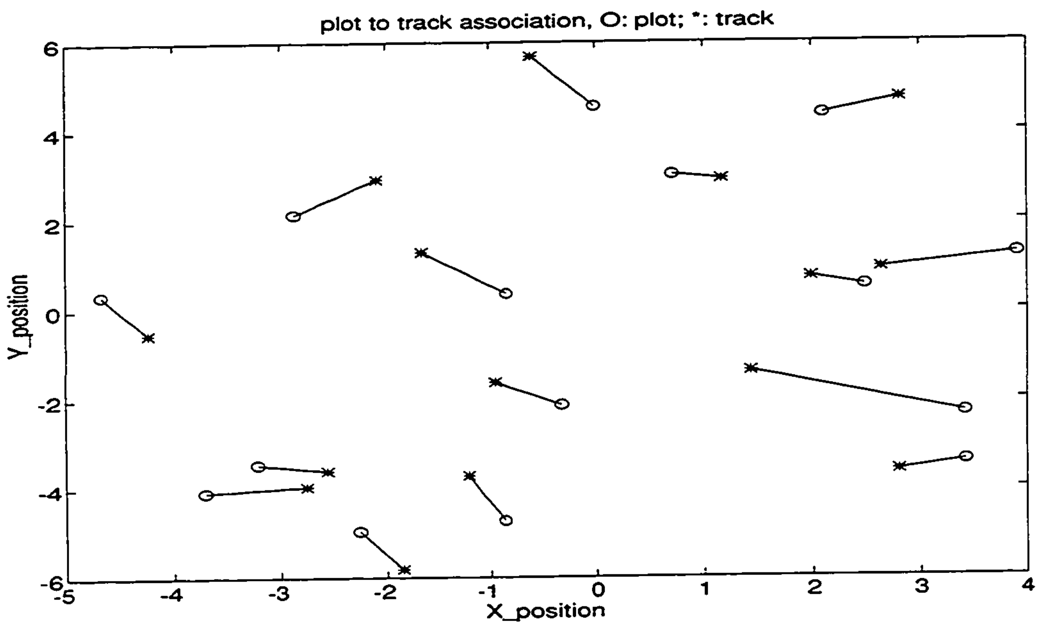


Figure 4.4: Association result for 15 plot-track pairs.

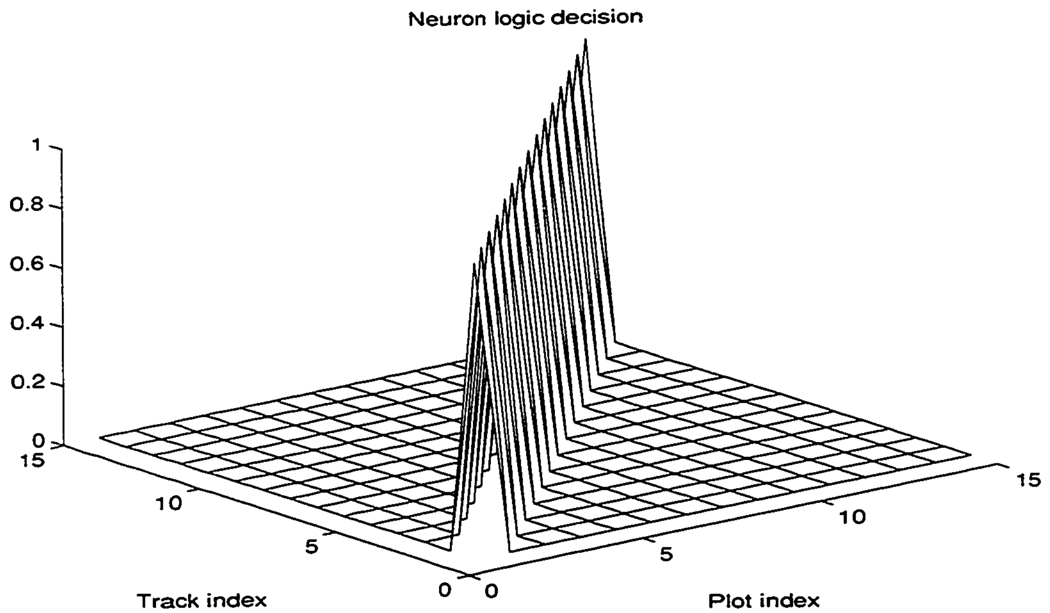


Figure 4.5: Neuron activation function signal for 15 plot-track pairs.

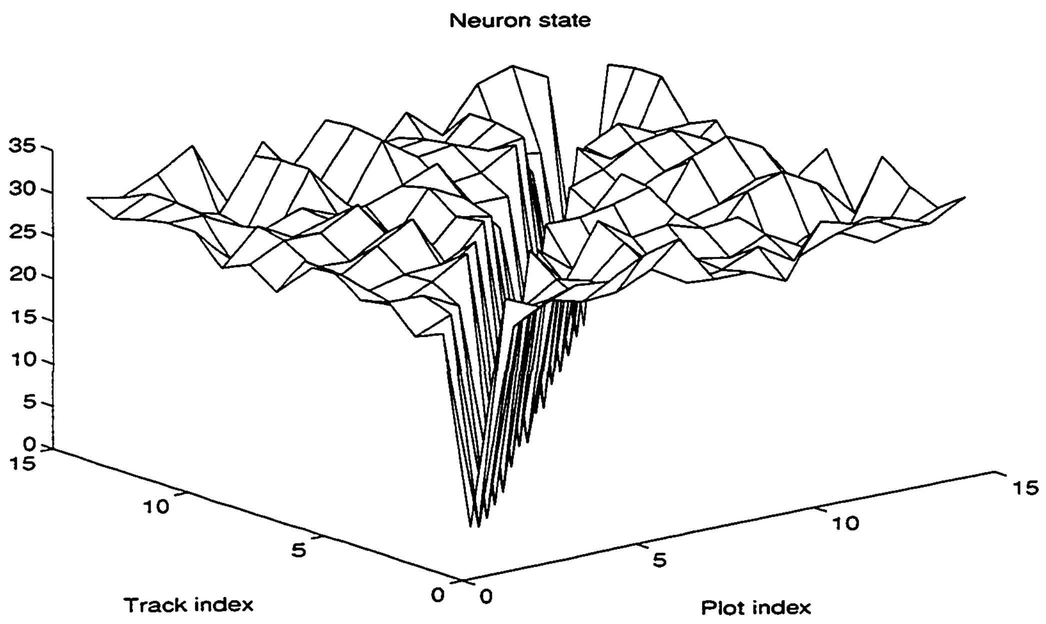


Figure 4.6: Neuron mean field state for 15 plot-track pairs.

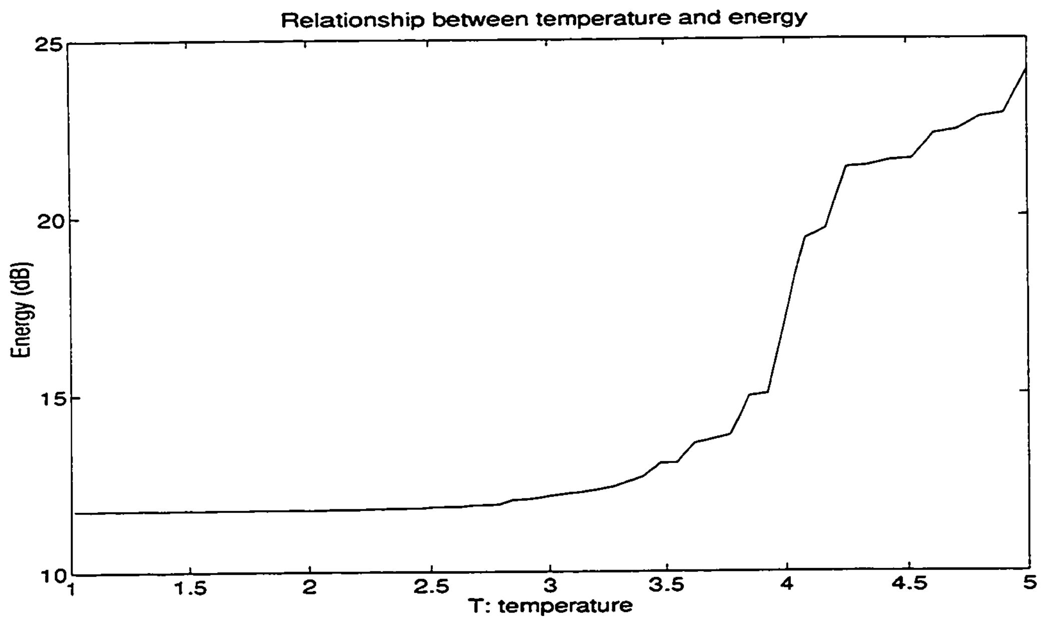


Figure 4.7: Energy variation versus simulated temperature for 15 plot-track pairs.

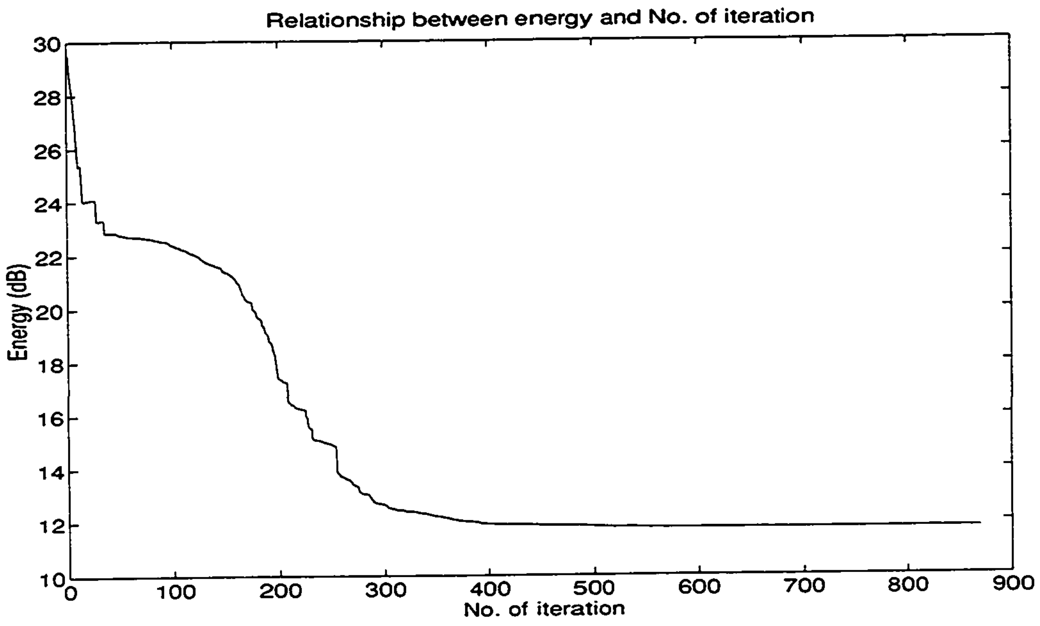


Figure 4.8: Energy variation versus No. of iteration for 15 plot-track pairs.

Based on a large number of experiments, we have found that there is a precipitous drop in E at certain temperatures. Continuing the annealing through temperatures much beyond the drop point is wasted work since it has little effect on the energy function; actually, most of the optimization occurs at a lower temperature, which is the so called critical temperature T_c. Carrying out the annealing at temperatures much less than the critical one does not qualitatively alter the solution found near T_c, but serves only to saturate the neurons at 1 or 0. Thus annealing is useful only over a certain range around the critical temperature. In practice, annealing will be carried out for temperatures below T_c to saturate the neurons in order to invoke assignment logic. Carrying out annealing around T_c will save on the computation overload. Now we start to find the critical temperature.

At a given simulated temperature, T, the energy function is a rough super-hyperboloid. At high temperatures in an n-pair plot-track case, each track will associate with each plot with a probability of approximately $\frac{1}{n}$. Suppose that plot X coagulates with track i , and the probability of plot Y associating track i changes by a small amount, $\triangle v_{Yi}$. From Eq. 4.6, we have

$$\triangle E_{Xi} = d_p \triangle v_{Yi}, \tag{4.8}$$

and from Eq.4.7, we derive

$$\frac{\triangle v_{Xi}}{\triangle E_{Xi}} = \frac{e^{-E_{Xi}/T}(-\frac{1}{T})(\sum_j e^{-E_{Xj}/T} - e^{-E_{Xi}/T})}{(\sum_j e^{-E_{Xj}/T})^2}, \tag{4.9}$$

and by simplifying, we obtain

$$\frac{\triangle v_{Xi}}{\triangle E_{Xi}} = v_{Xi}(v_{Xi} - 1)/T. \tag{4.10}$$

By combining the above formulas, we get

$$\triangle v_{Xi} = v_{Xi}(v_{Xi} - 1)/T \times d_p \triangle v_{Yi}. \tag{4.11}$$

For the critical temperature T_c, $\triangle v_{Yi} = - \triangle v_{Xi}$, i.e., when plot X coagulates on track i, any change of the neuron activation function signal, v_{Yi}, will be absorbed by the neuron activation function signal v_{Xi}. Before coagulation, the probability of each plot associating with each track is approximately $\frac{1}{n}$. Therefore we get roughly the critical temperature.

$$T_c = \frac{n-1}{n^2} d_p. \tag{4.12}$$

During experimentation, the annealing temperature is changed from large to small. At the beginning, the annealing temperature has to be larger than T_c by an amount $\triangle T$. In Fig.4.7, where $T_c = 3$, we can see that when the annealing temperature is sufficiently small, the slope of the energy is close to zero. Therefore, the neurons are in a stable state, which allows a decision to be made.

4.3 Dynamics of Mean-Field-Theory Machine

The stability of a nonlinear dynamical system is a difficult issue to deal with. When we speak of the stability problem, those of us with an engineering background usually think in terms of the bounded input-bounded output (BIBO) stability criterion. According to this criterion, stability means that the output of a system must not grow without bound as a result of a bounded input, initial condition, or unwanted disturbance. The BIBO stability criterion is well suitable for a linear dynamical system. However, it is useless to apply it to neural networks, simply because all such nonlinear dynamical systems are BIBO stable because of the saturating nonlinearity built into the constitution of a neuron.

The state of neural network is defined by a set of state variables whose values are neuron outputs. When the neural network evolves dynamically, a sequence of its states forms a trajectory. For investigating a nonlinear dynamical system with an equilibrium state $\overline{\mathbf{x}}$, the definitions of stability and convergence are as follows:

Definition 1. The equilibrium state $\overline{\mathbf{x}}$ is said to be uniformly stable if for any given positive ε, there exists a positive δ such that the condition

$$\|\mathbf{x}(0) - \overline{\mathbf{x}}\| < \delta$$

implies

$$\|\mathbf{x}(t) - \overline{\mathbf{x}}\| < \varepsilon$$

for all $t > 0$.

This definition states that a trajectory of the system can be made to stay within a small neighbourhood of the equilibrium state $\overline{\mathbf{x}}$ if the initial state $\mathbf{x}(0)$ is close to $\overline{\mathbf{x}}$.

Definition 2. The equilibrium state $\overline{\mathbf{x}}$ is said to be convergent if there exists a positive δ such that the condition

$$\|\mathbf{x}(0) - \overline{\mathbf{x}}\| < \delta$$

implies that

$$\mathbf{x}(t) \to \overline{\mathbf{x}} \quad \text{as} \quad t \to \infty.$$

The meaning of this second definition is that if the initial state $\mathbf{x}(0)$ of a trajectory is close enough to the equilibrium state $\overline{\mathbf{x}}$, then the trajectory described by the state vector $\mathbf{x}(t)$ will approach $\overline{\mathbf{x}}$ as time t approaches infinity.

Definition 3. The equilibrium state $\overline{\mathbf{x}}$ is said to be asymptotically stable if it is both stable and convergent.

Here we note that stability and convergence are independent properties. It is only when both properties are satisfied that we have asymptotic stability.

Definition 4. The equilibrium state $\overline{\mathbf{x}}$ is said to be asymptotically stable in the large, or globally asymptotically stable if it is stable and all trajectories of the system converge to $\overline{\mathbf{x}}$ as time t approaches infinity. At the same time, the trajectories of this kind of system (described by an energy function) are also called globally asymptotically stable.

Clearly, this definition implies that the system cannot have other equilibrium states, and it requires that every trajectory of the system remains bounded for all time $t > 0$. In other words, globally asymptotic stability implies that the system will ultimately settle down to a steady state for any choice of initial conditions.

The state of the neural network is described by a set of state variables. For convenience, we suppose that $\mathbf{x}_o$ is the state of the neural network such that at any moment and for any initial state, the condition $E(m) \geq E_o$, is satisfied, where E_o is the neural network energy corresponding to the state $\mathbf{x}_o$. We define $\mathbf{x}_o$ as a global minimization state, and E_o as a global minimum energy. When the temperature changes, the state of neural network forms a trajectory. If $\lim_{m\rightarrow\infty} E(m) = E_o$ for any initial state, the trajectory of the energy function $E(m)$ is called globally asymptotically stable. If an energy function of the mean-field-theory machine has a global minimum state, we have the following theorem:

Theorem: *The trajectories of a mean-field-theory machine will be globally asymptotically stable when using simulated temperature annealing and neuron normalization, under the condition that the neurons interact by the mean fields, and the* *outputs of the neurons are proportional to Boltzmann distribution.*

This theorem provides an affirmative assertion that the mean-field-theory machine globally asymptotically converges to a stable equilibrium provided the network satisfies the conditions described in the theorem. It also gives us a design rule.

Proof:

It is supposed that T_0 is a temperature at which the network arrives at an equilibrium, later we will show that it is a global optimization point. Since $T \rightarrow T_0$, $\triangle T < 0$. To prove the theorem, it is required to show that $\frac{dE}{dT} > 0$. If we assume that simulated annealing is applied to the network, while satisfying the condition $\frac{dE}{dT} > 0$, the energy of neural network will change, and satisfy:

$$\triangle E = \frac{dE}{dT} \triangle T < 0.$$

Because $E(m+1) = E(m) + \triangle E$, we get $E(m+1) < E(m)$. Combining this with the fact that the energy function is bounded, we conclude that the network converges to a stable state, that is, $\lim_{m-\infty} E(m) \rightarrow E_o$. Therefore E_o is a global minimum energy, otherwise the energy function would continue to decrease.

Based on the condition that the outputs of neurons are proportional to Boltzmann distribution, and that the neurons are connected to one another by the mean fields, and are normalized, $\frac{dE}{dT}$ is derived as follows.

$$\begin{aligned}
\frac{dE}{dT} &= \sum_i \frac{\partial E}{\partial v_{Xi}} \frac{\partial v_{Xi}}{\partial T} \\
&= \sum_i E_{Xi} \left(\frac{e^{-E_{Xi}/T}}{\sum_j e^{-E_{Xj}/T}} \right)'
\end{aligned}$$

$$
\begin{aligned}
&= \sum_i E_{Xi}\left[\frac{e^{-E_{Xi}/T}E_{Xi}/T^2\sum_j e^{-E_{Xj}/T} - (\sum_j e^{-E_{Xj}/T}E_{Xj}/T^2)e^{-E_{Xi}/T}}{(\sum_j e^{-E_{Xj}/T})^2}\right] \\
&= \sum_i E_{Xi}\frac{e^{-E_{Xi}/T}}{T^2(\sum_j e^{-E_{Xj}/T})^2}\left[E_{Xi}\sum_j e^{-E_{Xj}/T} - \sum_j e^{-E_{Xj}/T}E_{Xj}\right] \\
&= \sum_i E_{Xi}\frac{v_{Xi}}{T^2(\sum_j e^{-E_{Xj}/T})}\left[E_{Xi}\sum_j e^{-E_{Xj}/T} - \sum_j e^{-E_{Xj}/T}E_{Xj}\right] \\
&= \sum_i E_{Xi}v_{Xi}\frac{1}{T^2}\left[E_{Xi}\sum_j v_{Xj} - \sum_j v_{Xj}E_{Xj}\right] \\
&= \frac{1}{T^2}\left[\sum_i E_{Xi}v_{Xi}\sum_j v_{Xj}E_{Xi} - \sum_i E_{Xi}v_{Xi}\sum_j v_{Xj}E_{Xj}\right] \\
&= \frac{1}{T^2}\left[\sum_i\sum_j v_{Xi}v_{Xj}E_{Xi}^2 - \sum_i\sum_j E_{Xi}E_{Xj}v_{Xi}v_{Xj}\right]. \qquad (4.13)
\end{aligned}
$$

Due to neuron normalization, $\sum_j v_{Xj} = 1$, therefore the equation above can be rewritten:

$$
\begin{aligned}
\frac{dE}{dT} &= \frac{1}{T^2}\left[\sum_i v_{Xi}E_{Xi}^2 - \sum_i\sum_j v_{Xi}v_{Xj}E_{Xi}E_{Xj}\right] \\
&= \frac{1}{T^2}\left[\sum_i v_{Xi}E_{Xi}^2 - \sum_i v_{Xi}E_{Xi}\sum_j v_{Xj}E_{Xj}\right] \\
&= \frac{1}{T^2}\left[\sum_i v_{Xi}E_{Xi}^2 - (\sum_i v_{Xi}E_{Xi})^2\right]. \qquad (4.14)
\end{aligned}
$$

Let $\sqrt{v_{Xi}} = a_i$, and $\sqrt{v_{Xi}}E_{Xi} = b_i$, by the use of the inequality:

$$(\sum_i a_i b_i)^2 \le \sum_i a_i^2 \sum_i b_i^2,$$

we can obtain:

$$(\sum_i \sqrt{v_{Xi}}\sqrt{v_{Xi}}E_{Xi})^2 \le \sum_i v_{Xi}\sum_i v_{Xi}E_{Xi}^2 = \sum_i v_{Xi}E_{Xi}^2.$$

In the above formula, if $E_{X1} = E_{X2} = ... = E_{Xn}$, equality holds. This will not happen, since E_{Xi} is the mean field which is determined by neuron states (outputs) and physical features of the problem to be optimized; the outputs of neurons are proportional to Boltzmann distribution. Consequently, we have

$$(\sum_i v_{Xi} E_{Xi})^2 < \sum_i v_{Xi} E_{Xi}^2.$$

By substituting the above inequality into Eq.4.14. we arrive at

$$\frac{dE}{dT} > 0.$$

In the above proof. we do not need to know the practical energy function E. E_{Xi} is the only quantity we deploy. Therefore that is ubiquitous in mean-field-theory machine. The energy function E depends on a physical problem. In Fig.4.7 is presented the variation of the energy function versus annealing temperature for a 15-pair plot-track association, where $T_c = 3$. We can see that when the annealing temperature is sufficiently small, the energy is constant, and the neurons are in their stable states. thereby allowing for a decision to be made.

4.4 Data Association Performance

In this subsection, we test the data association ability of the mean field Hopfield network by carrying out computer simulations. We describe the results of the performance analyses for two cases: those without clutter (false measurements) and those with clutter. The association results in various clutter environments are demonstrated. The comparison between the neural data association and a conventional method of nearest neighbour association is made. The tracking trials based on the neural data associator are included. The applicability of the neural network in the tracking systems is discussed.

4.4.1 Performance Evaluation

In the computer simulations, the radar plan position indicator (PPI) is normalized, the X-axis goes from -5 to 5 and Y-axis covers the same range. The track positions are distributed uniformly and plot position departures from the tracks are dominated by Gaussian distributions. The clutters are uniformly scattered on the screen. Each case (with certain numbers of plots, tracks and clutters) is run a hundred and fifty times to calculate the percentage of correct associations.

Figs.4.9, 4.10, 4.11 and 4.12 are the association ability curves plotted against the number of plot-track pairs for the cases with and without clutter appearing during each scan. The normalized measurement precisions are 0.15, 0.20, 0.25, and 0.3, respectively. In these trials (one hundred and fifty iterations each), the test result is counted as a correct association, if and only if all the plot-track pairs are assigned correctly. It can be seen that when target density is increased, the association capacity diminishes, and the association capacity for the case without clutter is always higher than that for the case with clutter. When the radar accuracy decreases, there are more chances for the clutter to be linked to the track positions and to form a misassociation. Therefore, the correct association ability degrades in the case with clutter. Comparing Figs.4.9-4.12, we can see that the above viewpoint is correct. In

Fig.4.12, the association ability curve goes down quickly as the number of plot-track pairs increases. It corresponds to the situation where the radar range is 10km, and the area into which the plot falls with a confidence of 99.7% is 3.6km. This kind of radar is rarely used in practice, since it has very poor accuracy. In the case without clutter, the error of plot-track association is due to the fact that the tracks fall near one another. Again in the case, when radar accuracy decreases, misassociations increase. This conclusion is highlighted by Fig.4.13 in which data association capacities of four cases corresponding to the situations without clutter are plotted. If we fix the track positions, and the separations are large enough, then the correct association percentage rises to 100%.

In Fig.4.14 is shown the neural data association ability in the case where various numbers of clutter returns are present. These simulations are based on four tracks. It can be seen that when the accuracy of measurement radar decreases, the association capacity is degraded. When the clutter density increases, especially in the most heavy clutter environment, the association ability deteriorates. This will be overcome by incorporating the target attributes (speed feature, RCS) into the neural network energy function. Actually, in the case of very heavy clutters, without target identification information, it is not possible to get highly accurate association capacity.

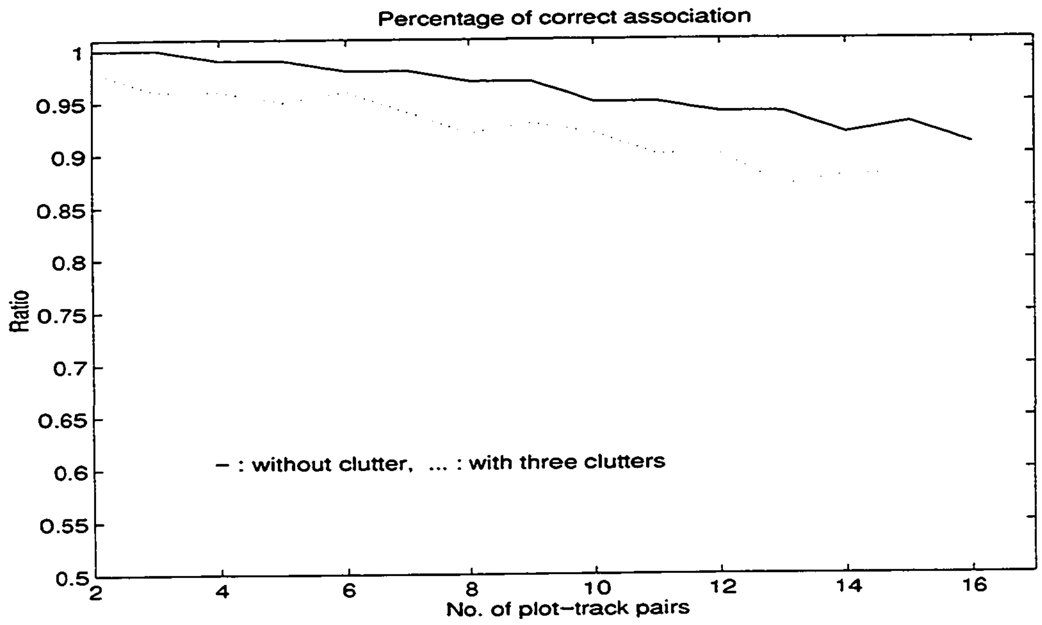


Figure 4.9: Percentage of correct association in the cases without clutter and with clutter. sensor precision: $\sigma_p = 0.15$.

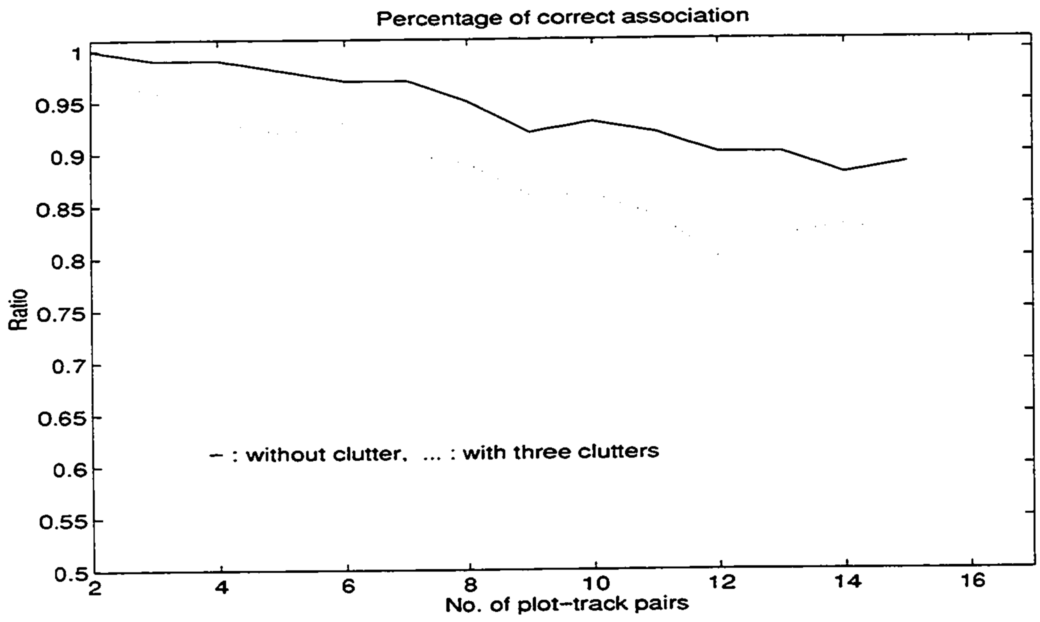


Figure 4.10: Percentage of correct association in the cases without clutter and with clutter, sensor precision: $\sigma_p = 0.20$.

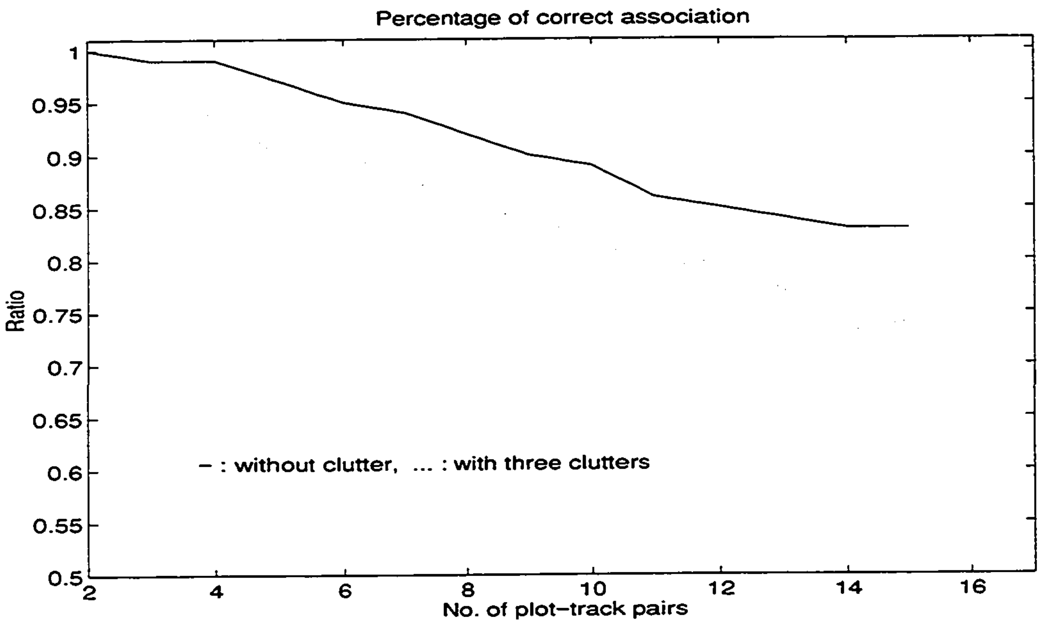


Figure 4.11: Percentage of correct association in the cases without clutter and with clutter, sensor precision: $\sigma_p = 0.25$.

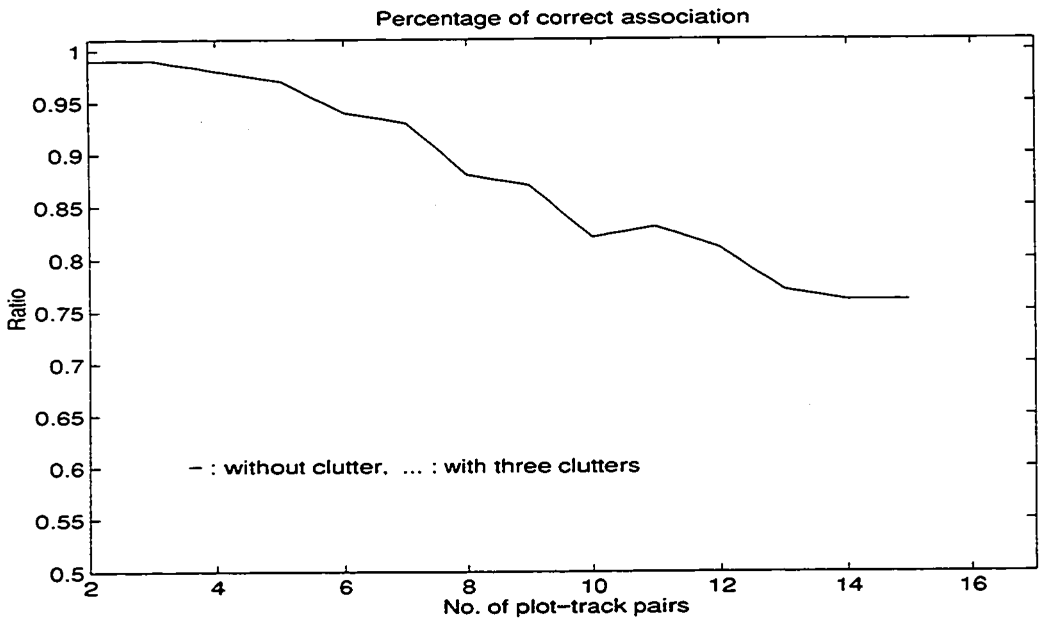


Figure 4.12: Percentage of correct association in the cases without clutter and with clutter, sensor precision: $\sigma_p = 0.3$.

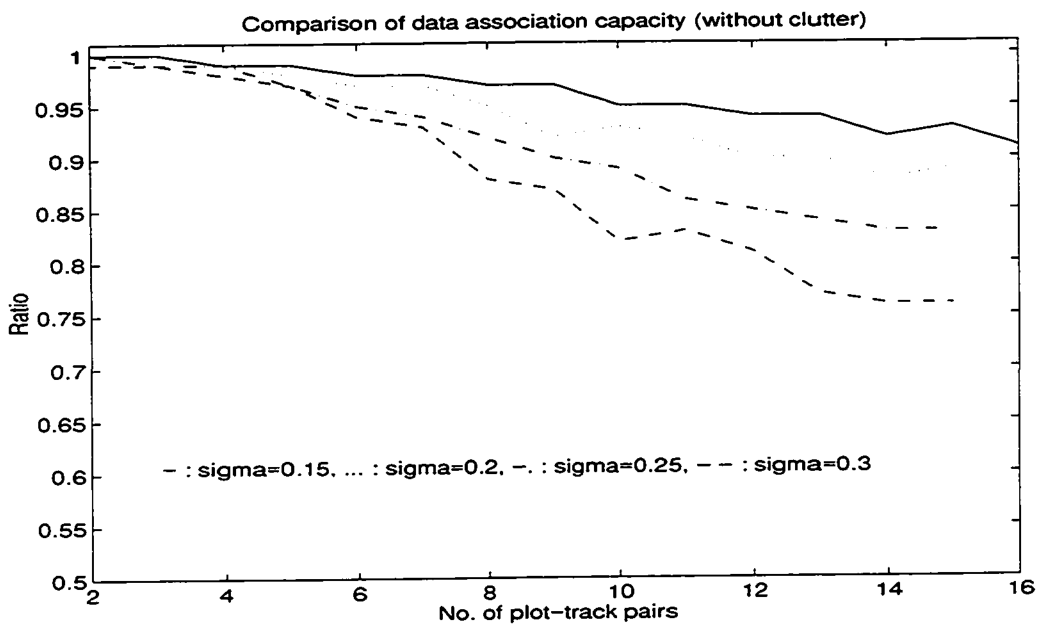


Figure 4.13: Percentage of correct association in the cases without clutter.

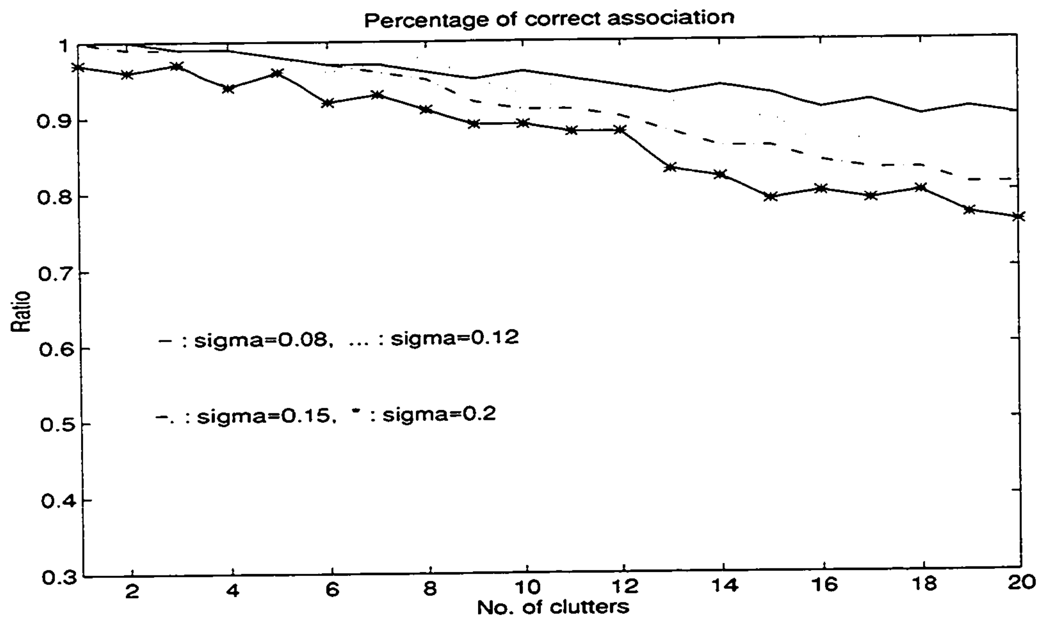


Figure 4.14: Data association capacity with different numbers of clutters.

4.4.2 Comparison of Neural and Conventional Data Association

This section makes a comparison between neural data association and nearest neighbour approach by computer simulations. The nearest neighbour association is a popular method for use in tracking systems. It works in such a way that a predicted point pairs with the closest radar measurement according to a distance norm. In the comparison, the PPI is normalized, both the X-axis and the Y-axis go from -5 to 5. The track positions are distributed uniformly in each trial, and plot position departures from the tracks are generated using a Gaussian random distribution. Each case, comprised of a certain number of plots and tracks, is run through a hundred and fifty trials to compute the percentage of correct associations for both methods, i.e., the neural data association and the nearest neighbour association. For the following results presented in this section, the test result is counted as a correct association, if and only if all the plot-track pairs are assigned correctly. In Fig.4.15 is shown the nearest neighbour association ability for four cases without clutter, in which the normalized measurement precisions (standard deviations) are 0.05, 0.15, 0.3 and 0.4, respectively. In Fig.4.16 is presented the comparison between the neural data association and the nearest neighbour association in the scenario without clutter, where the normalized measurement precision is 0.15. It can be seen that when the number of plot-track pairs is less than 11, the two methods have somewhat the same association ability, whereas, when the number of plot-track pairs increases, the neural data association has better association ability than does the nearest neighbour. In Fig.4.17 is given the comparison with the same situation as in Fig.4.16 except for the measurement precision being equal to 0.3. If in the previous case, the results did not clearly demonstrate the superior association ability of the neural network, the results in this case clearly show it. For 15 plot-track pairs, the neural data association capacity is about 20% higher than that for the nearest neighbour. It is a gain we obtained by using the combinatorial optimization network. Actually, the neural data association has more advantages than those presented here. In the next chapter, we demonstrate that neural data association has a powerful potential since it can deploy target attributes to associate plots and tracks.

It should be noted that from the viewpoint of engineering implementation, the nearest neighbour is easier to implement than the combinatorial optimization network. Therefore, the higher performance of neural network is achieved at the expense of the complexity of equipment structure. Fortunately, the rapid development of VLSI provides the opportunity for realizing complex networks.

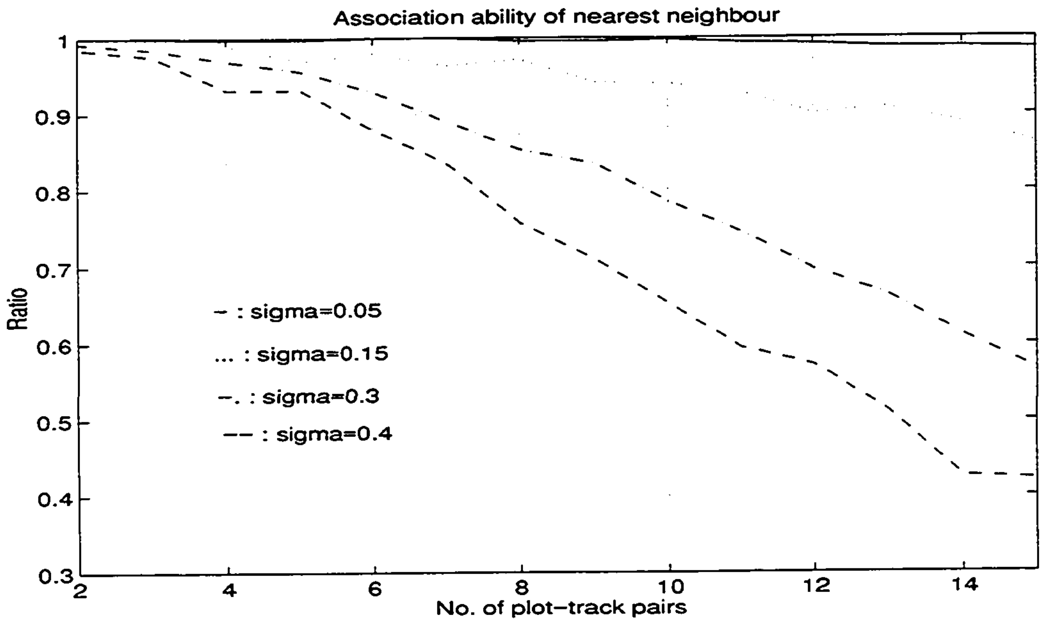


Figure 4.15: Percentage of correct association by using nearest neighbour in the cases without clutter.

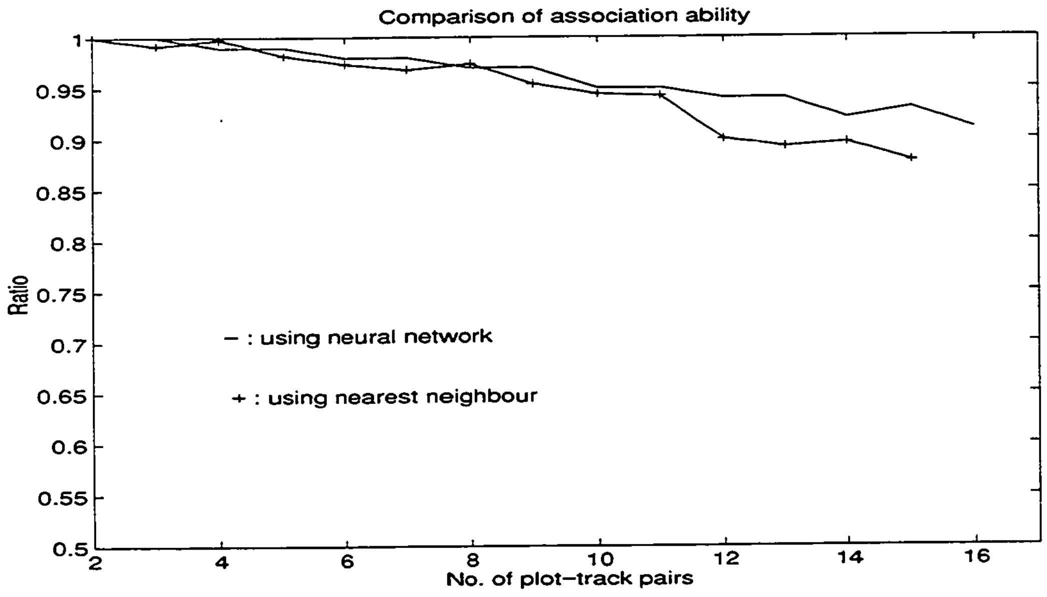


Figure 4.16: Comparison between the neural data association and the nearest neighbour method. $\sigma_p = 0.15$.

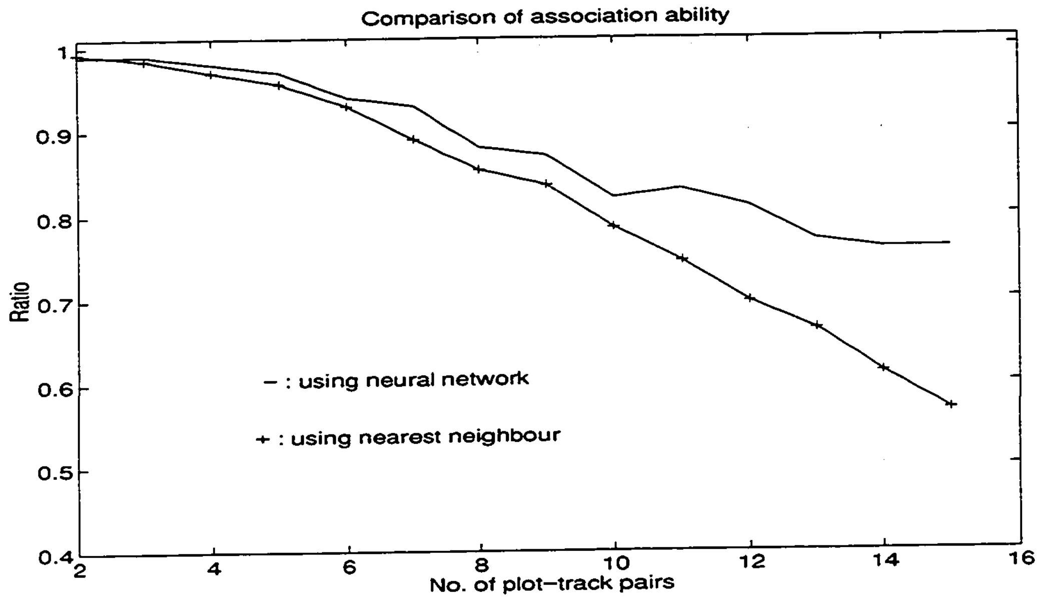


Figure 4.17: Comparison between the neural data association and the nearest neighbour method, $\sigma_p = 0.3$.

4.4.3 Applicability

To demonstrate the applicability of neural data association, we applied this technique to tracking. In order to show association, we do not use a filter (either a $\alpha - \beta$ filter or Kalman filter) to smooth the tracks. Fig.4.18 shows a simulated tracking result. In the trial, the measurement precision is chosen to be 0.8 and the screen is normalized as before. In Fig.4.19 is shown another tracking trial result in which the accuracy of measurement radar is 0.1.

In the aforementioned energy function, there is only one constant to be determined, which is simply set to a value larger than the twice the largest distance between any plot and track. On the other hand, referring to Fig.4.1, the neuron configuration, in combination with the energy function, annealing technique and normalization, is easily built into a VLSI neural network, which has a parallel computation power. Consequently, the technique presented here possesses potential applicability in MTT systems.

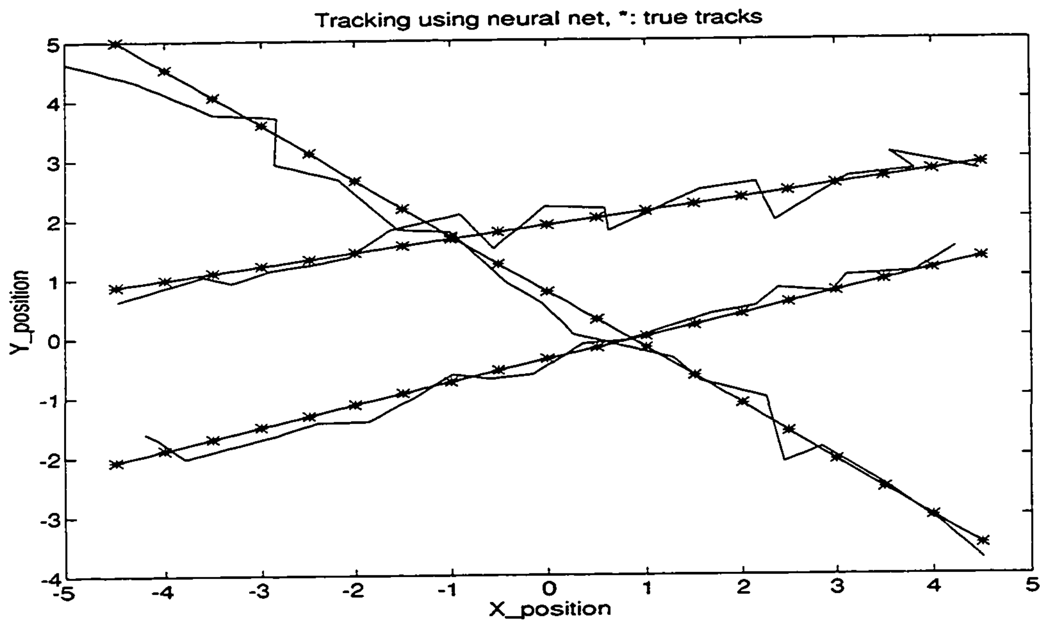


Figure 4.18: Tracking using neural data association.

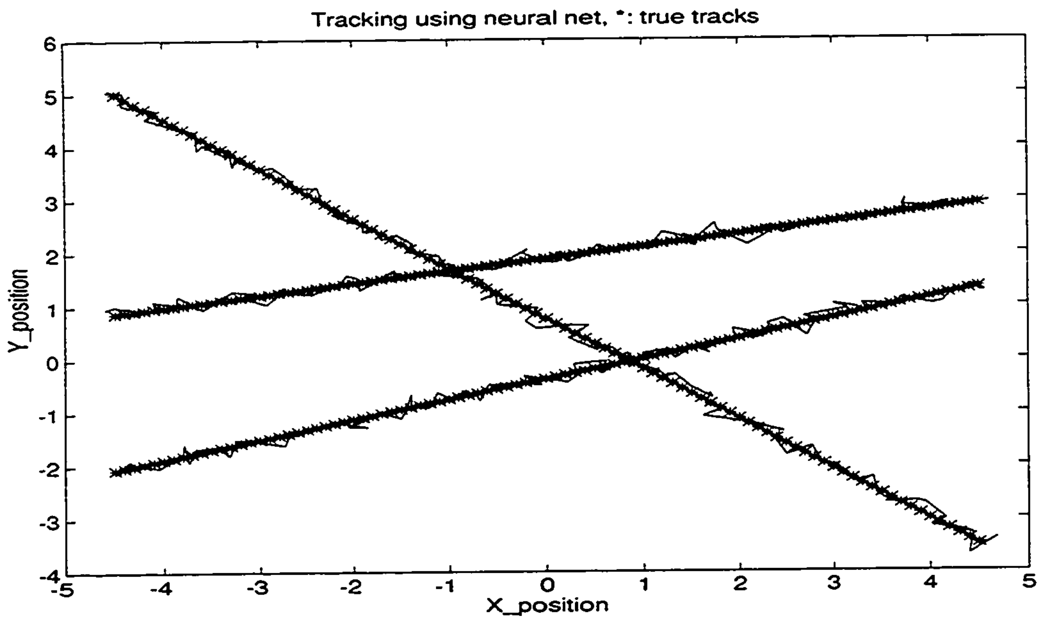


Figure 4.19: Tracking using neural data association.

4.5 Alternatives of Energy Functions for MDA

In the previous subsections, we have designed and analyzed an energy function for MDA. and demonstrated its capacity for implementing combinatorial optimization. As we mentioned before. many combinatorial optimization problems can be mapped onto neural networks by constructing suitable energy functions. thereby transforming the problem of minimization into one of associated networks. The formulation of the combinatorial energy function is the central issue in the design of optimizing neural networks. Based on the MDA problem and the paradigm built above, the other two new energy functions for MDA have been designed. In the following, the definitions of mathematical symbols are the same as in Eqs.4.4. 4.5 and 4.6. An alternative to the energy function can be expressed as

$$E = \frac{d_p}{2}\sum_i \sum_X \sum_{Y \neq X} v_{Xi}v_{Yi} + \sum_X \sum_i v_{Xi}d_{Xi}, \tag{4.15}$$

the mean field of a neuron. which corresponds to plot X and track i, can be obtained

$$E_{Xi} = d_p \sum_{Y \neq X} v_{Yi} + d_{Xi}. \tag{4.16}$$

The other alternative of the energy function can be represented by

$$E = \frac{d_p}{2}\sum_i \sum_X \sum_{Y \neq X} v_{Xi}v_{Yi} + \sum_X \sum_i v_{Xi}(\sum_{j \neq i} v_{Xj}d_{Xj} + d_{Xi}), \tag{4.17}$$

the mean field of a neuron describing plot X and track i is given.

$$E_{Xi} = d_p \sum_{Y \neq X} v_{Yi} + \sum_{j \neq i} v_{Xj}d_{Xj} + \sum_{j \neq i} v_{Xj}d_{Xi} + d_{Xi}. \tag{4.18}$$

The computational procedure of Eqs.4.15 and 4.17 follows the calculation procedure of Eq.4.4. Both energy functions can be used to obtain the results given in Figs.4.2 and 4.4, and get the global optimal data association in the different trials under the condition of setting initial neuron states in uniform distribution with standard deviation very small. It is found that Eq.4.15 is susceptible to the neuron initial states, i.e., if the neuron initial states are not set in uniform distribution with small deviation, sometime, the network output may not be optimal. Since E_{Xi} is more directly dominated by d_{Xi} than it is in Eq.4.17, the convergent speed of Eq.4.15 is faster than that of Eq.4.17. On the other hand, Eq. 4.17 is more tolerant of neuron initial setting than Eq.4.15. It can adjust when the annealing temperature goes down, but with the cost of lower speed. The convergent speed of Eq.4.4 is a compromise between Eq.4.15 and Eq.4.17.

4.6 Comments

A new technique of measurement data association in a multitarget radar environment has been presented in this chapter. The technique is based on the mean-field-theory machine and has the advantages of both the Hopfield network and the Boltzmann machine. The new energy function, used in combination with normalization and annealing, can obtain a globally optimal solution. As it is known, track quality mainly depends on the associator's performance, therefore the technique presented here can enhance the performance of tracking systems.

The convergence theorem presented in this chapter provides an affirmative assertion that the energy function of the mean-field-theory machine globally asymptotically converges to a minimum point. Usually the energy function is the cost function of a physical problem. The theorem also offers a network design rule. Theoretically and practically, the energy function presented here is a new approach to measurement data association.

The mechanism of plot-track association which is presented here is based on finding a global optimization, which is susceptible to sensor degradation. When the sensor precision diminishes, the performance of neural data association technique will degrade in the two situations of interest, i.e., with and without clutter. In the case where clutter is present, the capacity of neural data association diminishes as the clutter density increases, as well as when the sensor's precision decreases. In the next chapter it will be shown that this limitation can be overcome by incorporating the target's attributes into the energy function.

Chapter 5

NDA with Multiple Dimensional Measurements

5.1 Principle

The motivation for using multiple dimensional measurements (attributes) to link plots (radar contacts) and tracks together is that we intend to deploy the distinct attributes of targets in order to be better able to make distinctions between them. so that the data association can be made clearly. Practically. drawing a clear line of distinction between targets is not easy. since the dynamic properties of targets change with time. However. we can use as much information as we can to get rid of the association ambiguity. In order to put across this idea, let us suppose that there are two aircraft that are sufficiently close that they are separated by less than a beamwidth, but that they move with different speeds. In a coordinate system consisting of position (X-Y) and speed (V), each aircraft falls in a probabilistic region which is an ellipsoid. Since the speeds of the aircraft are different, the two ellipsoids do not overlap, therefore, the targets can be easily separated. However, if only position measurements are used, the region in which an aircraft falls is the projection of an ellipsoid onto X-Y plane, the overlapping of the two projections, corresponding to two ellipsoids, is complete. In this situation, it is not possible to separate these targets based only on position measurements.

In Pulse Doppler (PD) radar, antenna pointing direction provides the elevation and azimuth; range gates indicate the target distances, and the outputs of a Doppler filter bank provide target speeds. By intuition, in addition to using position measurements, we can extend our method to incorporate Doppler information. Generally, we can deploy multiple dimensional measurements. Suppose that we have n-dimensional measurements for each target, and denote them as $Z_X = (z_{X1}, z_{X2}, \dots, z_{Xn})^T$, $X = 1, \dots, K$, where X stands for the index of the radar contact. A predicted point is defined as $P_j = (p_{j1}, p_{j2}, \dots, p_{jn})^T$, $j = 1, \dots, M$. The norm distance between a predicted point and a measured point can be written as

$$d_{Xj} = \|Z_X - P_j\|^{1/2}, \quad X = 1, \dots, K, \quad j = 1, \dots, M. \tag{5.1}$$

The ranges of X and j are different due to the appearance of false alarms (clutters). Again, the multiple dimensional data association issue can be translated into a combinatorial optimization problem. All the developments of an optimization network are similar to that in Chapter 4. For convenience, we reproduce the energy function here, i.e.

$$E = \frac{d_p}{2}\sum_i \sum_X \sum_{Y \neq X} v_{Xi} v_{Yi} + \sum_X \sum_i v_{Xi} \sum_j v_{Xj} d_{Xj}, \tag{5.2}$$

where X, Y denote plot index and i ,j denote track index, v_{Xi} is the neuron output which represents the probability that plot X associates with track i. d_{Xj} is the distance between plot X and track j, and d_p is a penalty constant. Refer to Chapter 4 for a description of the computation of the energy function for measurement data association.

5.2 Results

It is well known that in data association, there are two situations which are tough to deal with: a) tracks that are very close to one another or tracks that cross one another. b) a plenty of false alarms. The following demonstrates the capacity of multiple dimensional data association to handle these circumstances. In computer simulations, four tracks are fixed to be in close proximity, the track positions are: $p_1 = [1.6, 2.4]$, $p_2 = [2.2.8]$, $p_3 = [2.4, 2.4]$, $p_4 = [2, 2]$. The separations of the tracks are less than radar measurement errors. Furthermore, the target speed deviation due to the measurement error and acceleration is supposed to belong to a Gaussian distribution. During each scan, the RCS of each of the targets is supposed to remain unchanged, and is equal to the mean of a Rayleigh distribution of RCS. We are only concerned with the measurement error, which belongs to Gaussian distribution. In Figs.5.1-5.4 are shown the data association results in the case without clutter .

Figs.5.5-5.10 are association results in clutter environments. In Figs.5.1 and 5.2 are shown the association results without using speed and using speed, respectively, where the position measurement precision σ_p is 0.8, and the speed measurement precision σ_v is 0.1. The normalized speeds of four targets are 1, 2, 3, and 4, respectively. In Fig.5.1, all associations are wrong, which corresponds to the case where only position is used. In Fig.5.2, all associations are correct by using a target attribute. In Figs.5.3 and 5.4 are presented the association results without using RCS and using RCS, respectively, where $\sigma_p = 0.8$, the RCS measurement precision σ_R is 0.2, the RCS's of the four targets are 3, 1, 2, and 3.5 m^2, respectively. The advantage of using RCS is clear. In the following results, clutter speeds are set to be less than 0.5. In Figs.5.5 and 5.6 are given the association results without using speed and using speed, respectively, in the case with 10 clutter returns, where $\sigma_p = 0.5$, $\sigma_v = 0.1$; the normalized speeds of four targets are 6, 9, 12, and 3, respectively. In Figs.5.7 and 5.8 are given the association results where speed is deployed and not deployed, respectively, in the case of 50 clutter returns, where $\sigma_p = 0.6$, $\sigma_v = 0.2$. The target speeds are the same as in Fig.5.6. In Figs.5.9 and 5.10 are demonstrated the association results without using RCS and using RCS, respectively, in the case with 30 clutter returns, where $\sigma_p = 0.6$, $\sigma_R = 0.2$. The RCS's of targets are 3, 2.8, 2, and 1 m^2, respectively. The RCS's of clutter returns are set to be less than 0.2. Clearly, the multiple dimensional data association demonstrates powerful association ability in both cases, without and with heavy clutter returns.

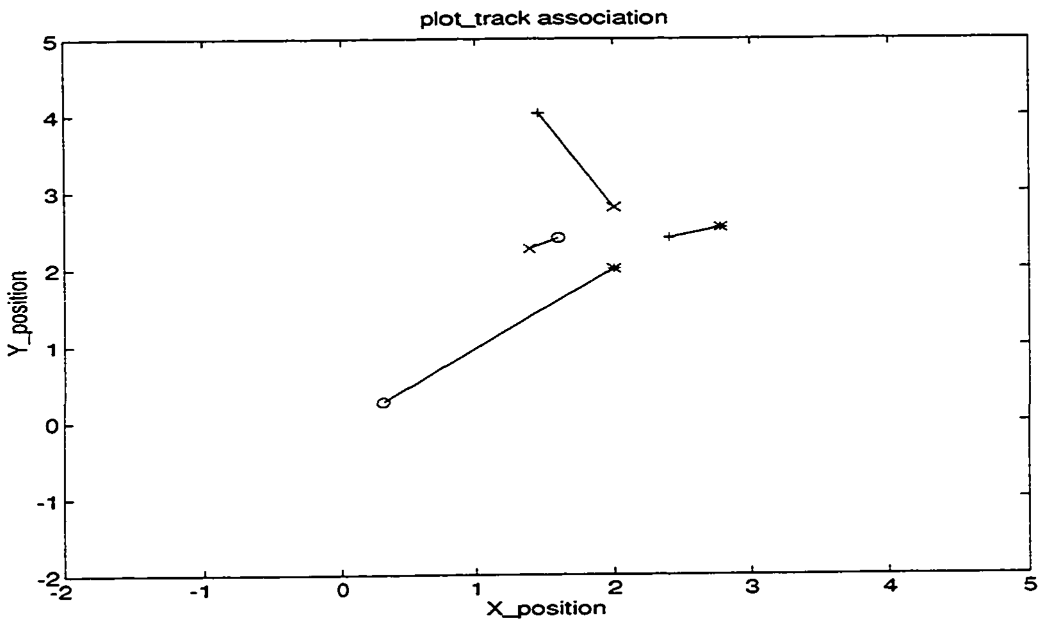


Figure 5.1: Data association using positions in the case without clutter . $\sigma_p = 0.8$.

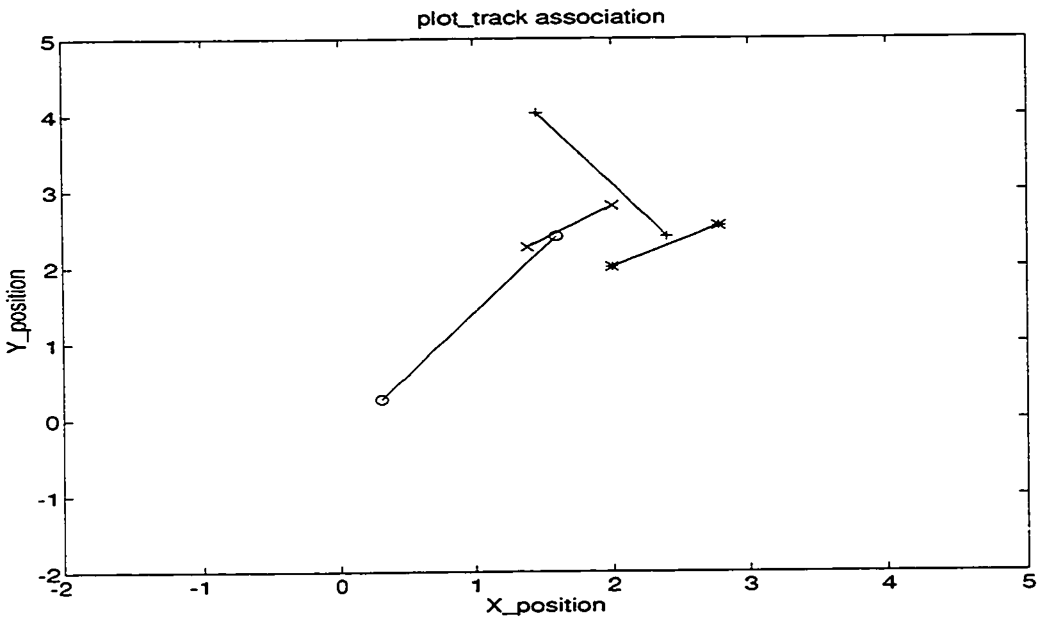


Figure 5.2: Data association using positions and speeds in the case without clutter , $\sigma_p = 0.8$.

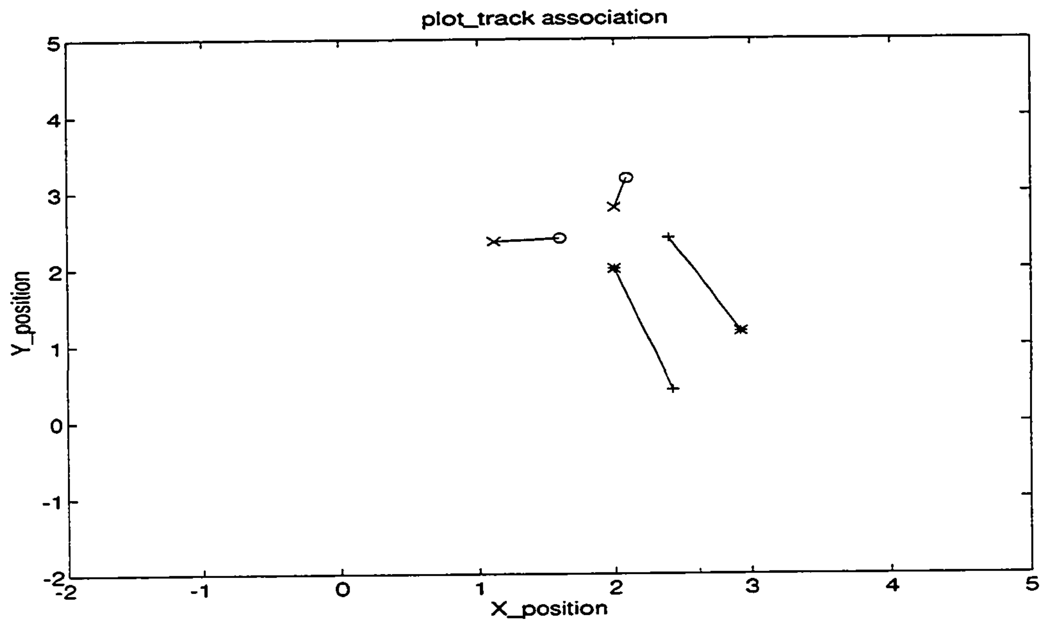


Figure 5.3: Data association using positions in the case without clutter , $\sigma_p = 0.8$.

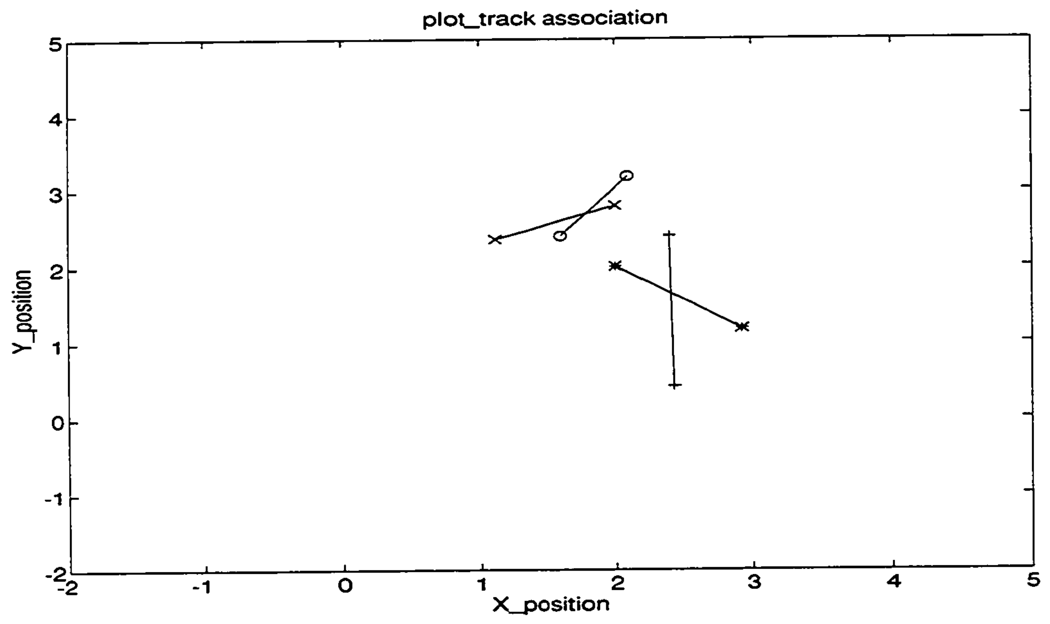


Figure 5.4: Data association using positions and RCS's in the case without clutter , $\sigma_p = 0.8$.

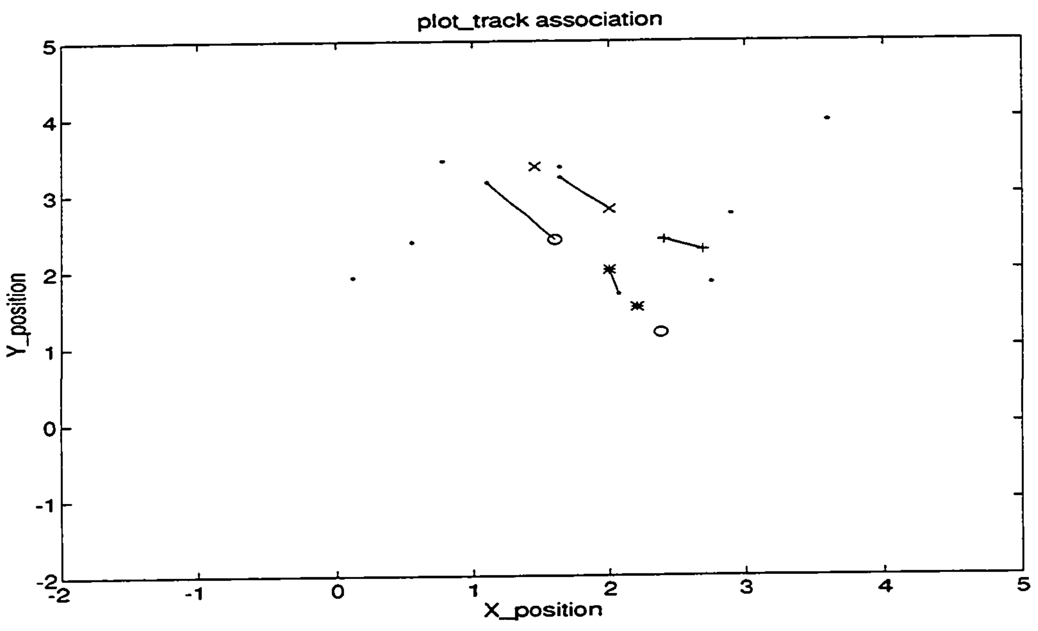


Figure 5.5: Data association using positions in the case with clutter returns. $\sigma_p = 0.5$.

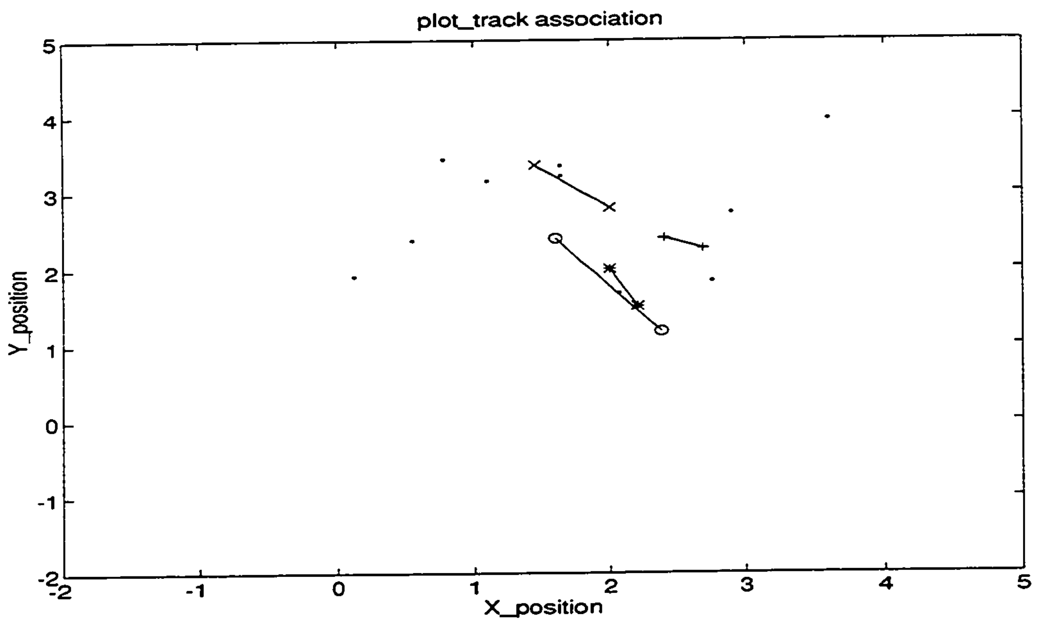


Figure 5.6: Data association using positions and speeds in the case with clutter returns. $\sigma_p = 0.5$.

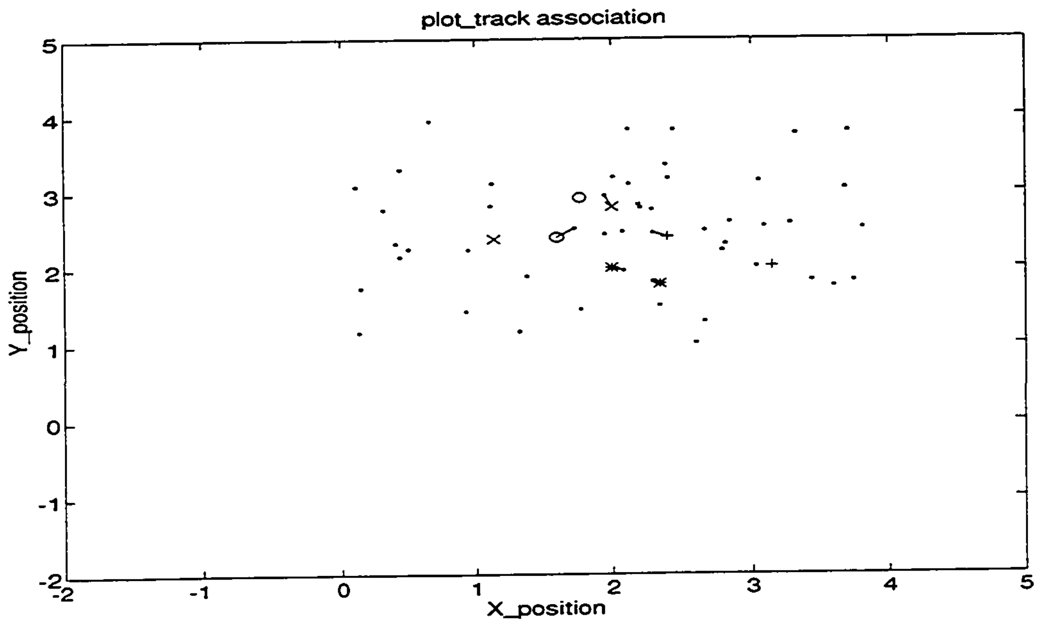


Figure 5.7: Data association using positions in the case with clutter returns. $\sigma_p = 0.6$.

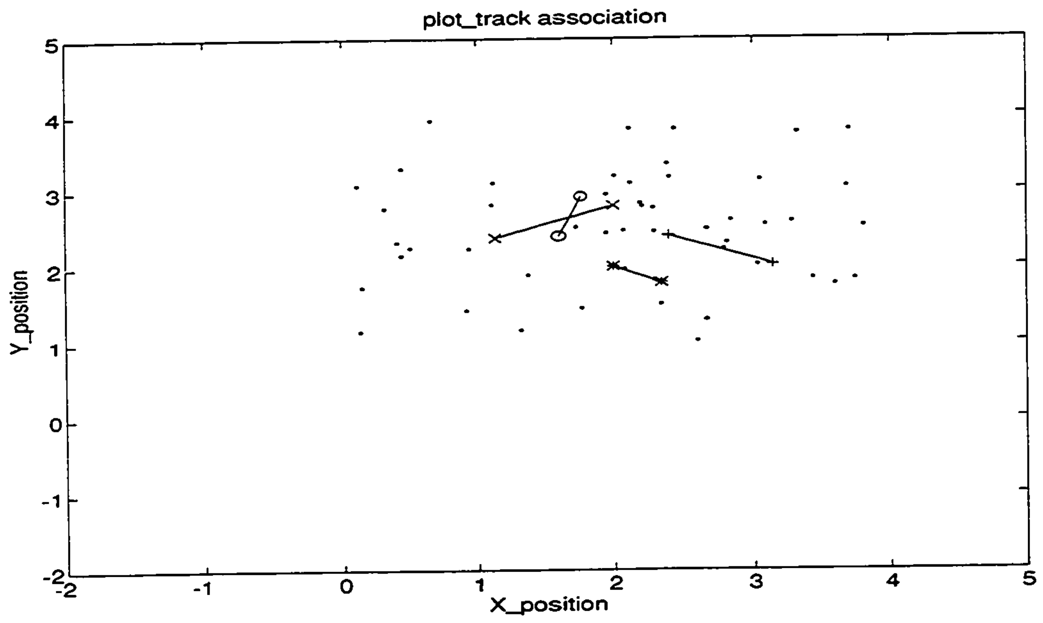


Figure 5.8: Data association using positions and speeds in the case with clutter returns, $\sigma_p = 0.6$.

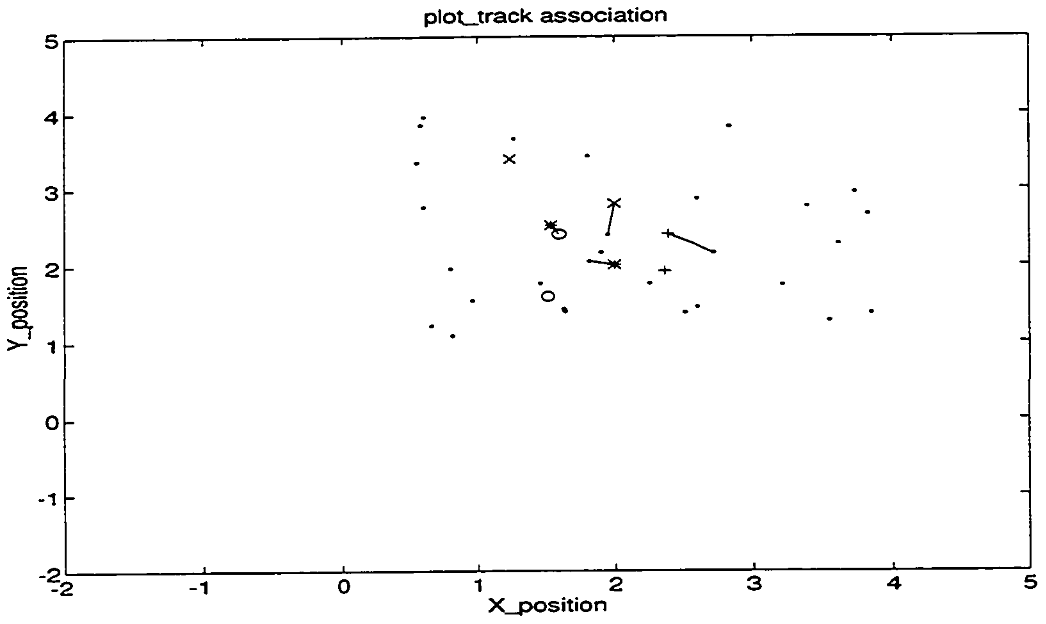


Figure 5.9: Data association using positions in the case with clutter returns. $\sigma_p = 0.6$.

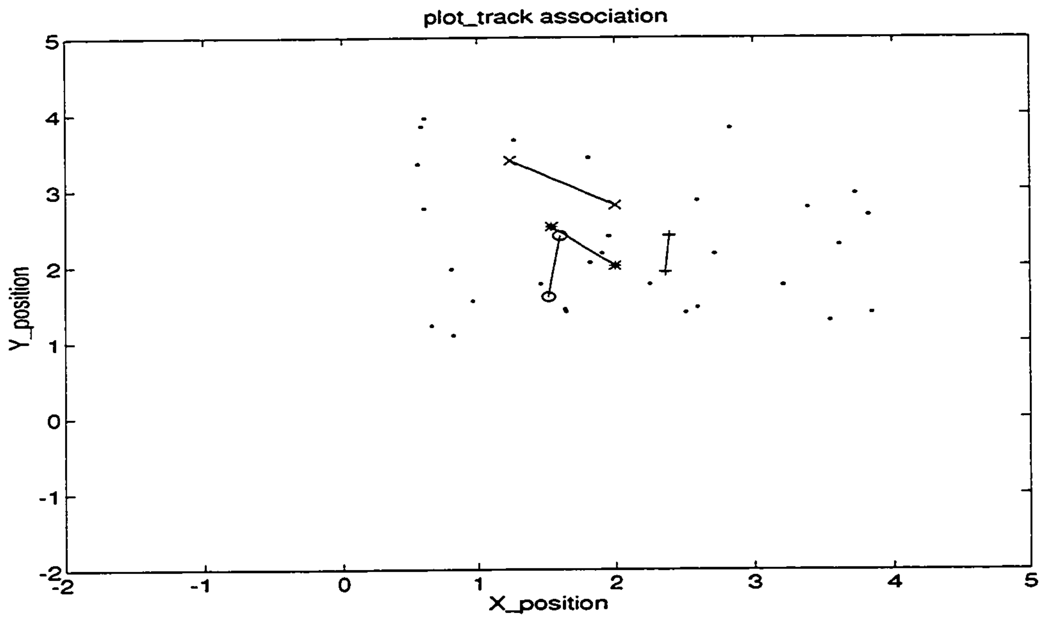


Figure 5.10: Data association using positions and RCS's in the case with clutter returns, $\sigma_p = 0.6$.

5.3 Comparison

In parallel with Chapter 4, further performance analyses have been carried out using computer simulations. In the following performance analysis, the PPI is normalized, and both the X-axis and the Y-axis go from -5 to 5. Unless otherwise stated, the track positions are distributed uniformly in each trial, and plot position departures from the tracks are generated from Gaussian random distribution. Each case, with certain numbers of plots and tracks, as well as speed and RCS parameters, is run a hundred and fifty times to compute the percentage of correct associations. For the following results presented in this section, the test result is counted as a correct association, if and only if all the plot-track pairs are assigned correctly. In Fig.5.11 is shown the association capacity for both using speed and not using speed in the case without clutter, where $\sigma_p = 0.2$, $\sigma_v = 0.2$, normalized target speeds are uniformly distributed in $[3 - 10]$. In Fig.5.12 is presented the association ability for the case where speed is and is not considered. Three clutter returns per scan are assumed, where $\sigma_p = 0.2$, $\sigma_v = 0.2$. Target speeds are uniformly distributed in the range of $[3 - 10]$, clutter speeds are uniformly distributed in the range of $[0 - 10]$. The reason we choose this range for the clutter speed is that, in addition to slow speed clutters generated by flocks of birds or mountains, the new targets which may enter into the illuminated zone, with predesignated speeds, will be considered as "clutter" for the existing tracks. Only after initiation and the formation of a new track, can the association start.

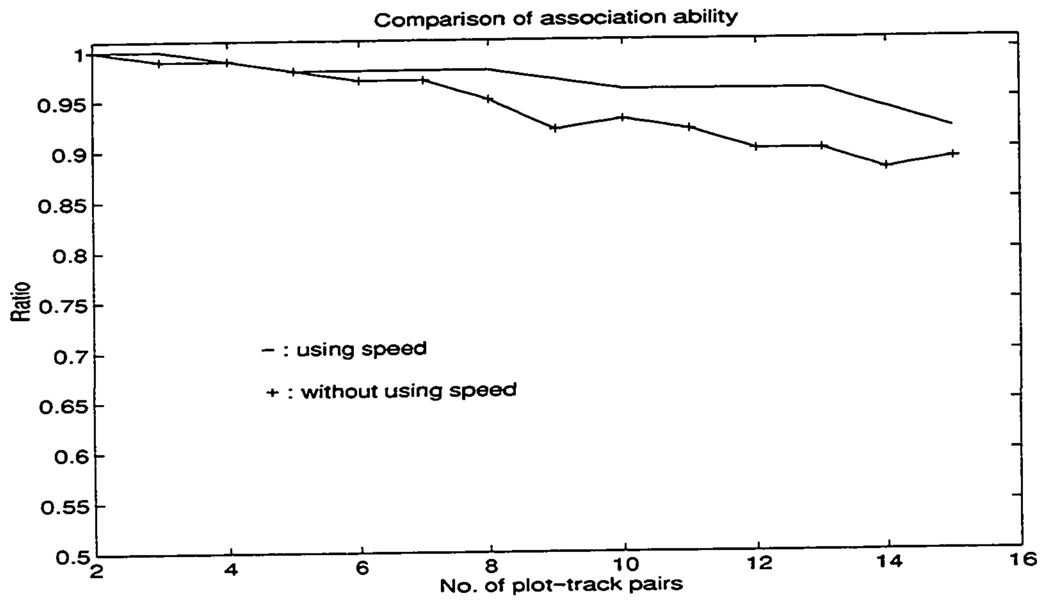


Figure 5.11: Comparison of data association between using speed and without using speed in the case without clutter , $\sigma_p = 0.2$.

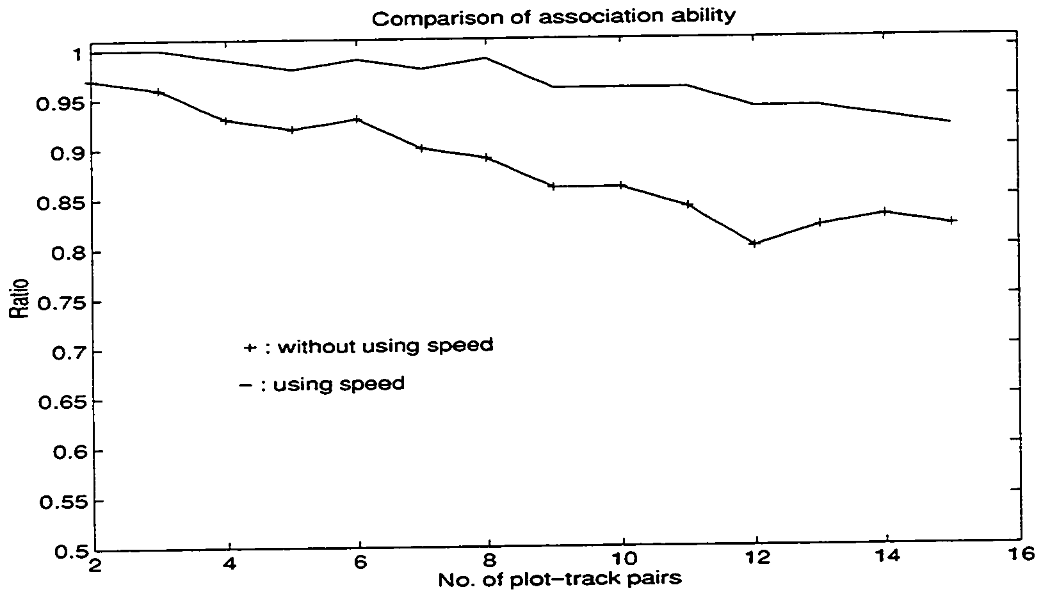


Figure 5.12: Comparison of data association between using speed and without using speed in the case with 3 clutter returns, $\sigma_p = 0.2$.

In Fig.5.13 is shown the association capacity for the case of four fixed tracks with increasing clutter densities, where $\sigma_p = 0.2$, $\sigma_v = 0.2$; the speed parameters are the same as in Fig.5.12. It can be seen that the association ability obtained by using speed information is 20% higher than that obtained without using speed information. In Fig.5.14 is given the performance comparison of the neural data association for three cases, 1) that based on position association, 2) that based on position and speed association, 3) the remainder, which is based on position, speed and RCS association. It is supposed that there are three clutter returns per scan, and $\sigma_p = 0.3$, $\sigma_v = 0.2$, $\sigma_R = 0.2$. The speed parameters are the same as in Fig.5.12. The means of the RCS's are in the range of $[3 - 5]$. It clearly demonstrates that the more information the optimization network incorporates, the higher the data association ability of the network. In this particular case, data association ability with multiple dimensional measurements is 30% higher than that only using position measurements. The comprehensive analysis carried in this section using computer simulations supports our motivation for using multiple dimensional measurements to enhance the data association ability.

By an analogy of the detection of a signal in a noise environment, we define an "association threshold" to assess the performance of the neural data association. First, we define a quantity ρ as a ratio of minimal track distance L_{min} and measurement radar precision σ:

$$\rho = \frac{L_{min}(\text{minimal distance})}{\sigma(\text{precision})} \tag{5.3}$$

The ρ_t, which corresponds to a 95% correct association, is defined as an association threshold. In computer simulations, four tracks are fixed closely in an environment without clutter . The track positions are $p_1 = [1.6, 2.4]$, $p_2 = [2, 2.8]$, $p_3 = [2.4, 2.4]$, $p_4 = [2, 2]$. At each ρ value, two hundred association trials are carried out for both neural associators which deploy positions only and positions and speeds, respectively. Again, each test result is counted as a correct association, if and only if all the plot-track pairs are assigned correctly. In Fig. 5.15 is presented the association performance versus ρ. According to the definition of association threshold, we can find that the threshold for the neural data associator using positions is 2.68, the counterpart for the neural data associator using positions and speeds is 1.2. It can be concluded that the more information the associator uses, the lower the association threshold is.

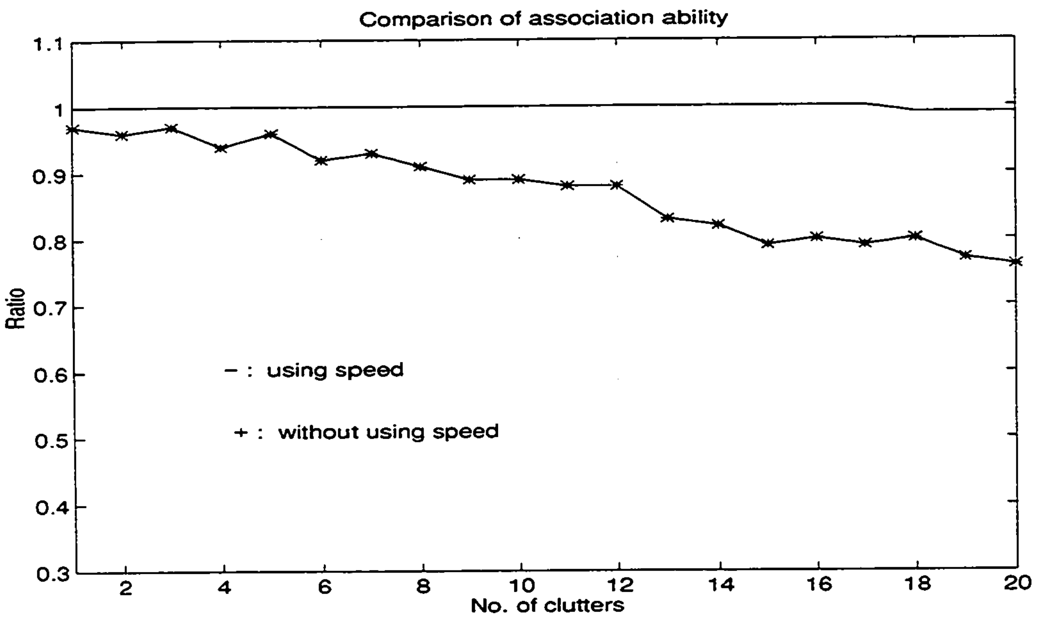


Figure 5.13: Comparison of the data association between using speed and without using speed in the cases of different clutter densities, $\sigma_p = 0.2$.

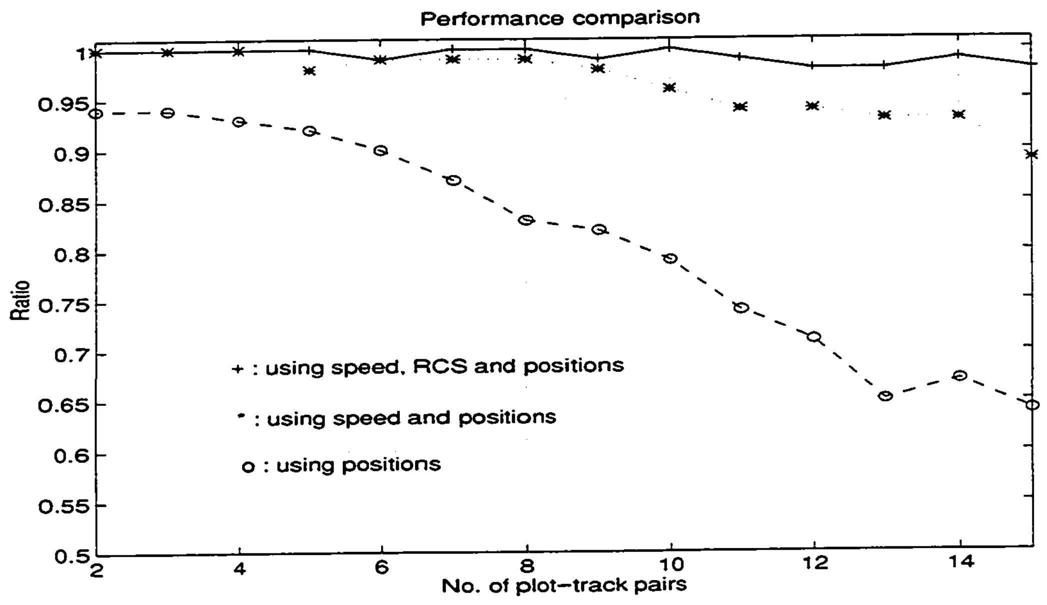


Figure 5.14: Comparison of the neural data association using multiple dimensional measurements, $\sigma_p = 0.3$.

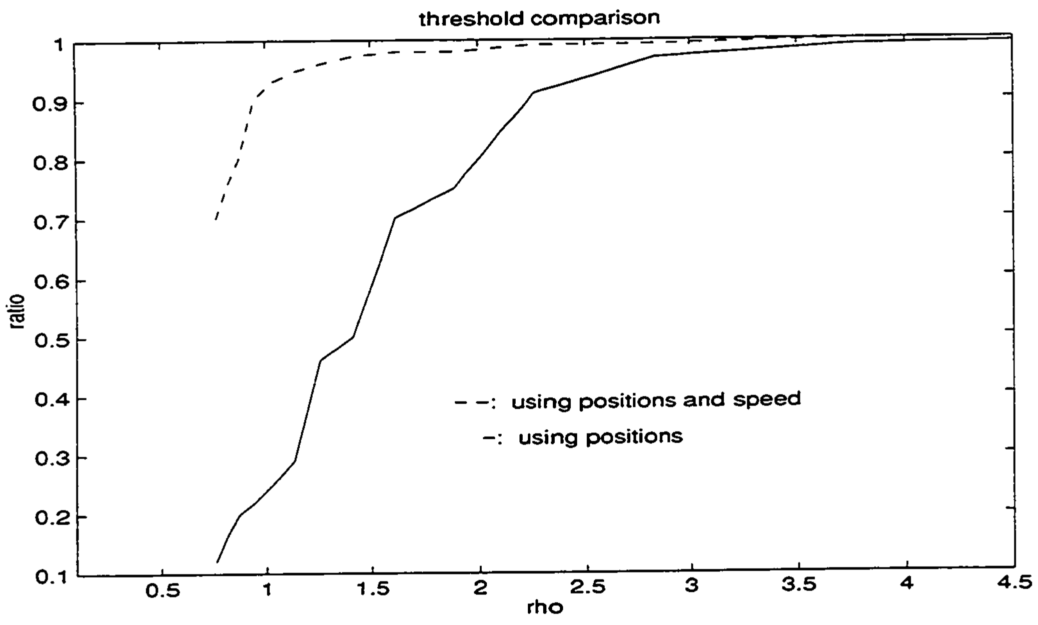


Figure 5.15: Data association performance versus ρ.

Chapter 6

Multisensor Target Classification

This chapter presents the multisensor data fusion for airborne target classification using an artificial neural network. A feature set contaminated by noise, which possesses the dominant characteristics of targets and has a certain dynamic range, is chosen to teach the network. The entire system consists of identification nets (IN) and classification net (CN). Like hierarchical structures of adaptive expert networks, it makes the learning process fast. In the system, each subsystem is a simple neural network. It is easy to train the neural network which has a simple structure. For one simple target signature, one hidden layer is enough to make the simple network converge. If uncorrelated signatures were mixed together to teach a simple network, the network would not converge. Even though we can increase the number of hidden layers to make the network converge, we find that the convergent speed is very slow due to the uncorrelated signatures. Each identification network is used to extract a particular feature of the target, then the outputs of identification networks are fused by a classification network to give the decision.

In this chapter, we discuss multilayer perceptrons neural networks trained by the back-propagation (BP) rule. In a practical application of the back-propagation algorithm, learning results from the many presentations of a prescribed set of training examples to the multilayer perceptrons. One complete presentation of the entire training set during the learning process is called an *epoch*. The learning process is maintained on an epoch-by-epoch basis until the synaptic weights and threshold levels of the network stabilize, and the average squared error over the entire training set converges to some minimum value. In order to speed up the training or decrease the number of epochs required in the learning process, both momentum and adaptive learning rate methods are used. The simulation results show that the technique of automatic target classification using multilayer perceptrons neural networks can achieve robust decision performance.

6.1 Multilayer Perceptrons

A multilayer perceptron consists of an input layer, hidden layers and an output layer. The input layer is constituted of source nodes, which has a function of distributing signals. The hidden layers and output layer consist of neurons, which perform a transformation function [50, 89].

A multilayer perceptron, as shown in Fig. 6.1 , has three distinctive characteristics: (1) The model of each neuron in the network includes a nonlinearity at the output. The important point to emphasize here is that the nonlinearity is smooth.

A commonly used form of nonlinearity that satisfies this requirement is a sigmoidal nonlinearity defined by the logistic function:

$$y_j = \frac{1}{1 + \exp(-v_j)}, \tag{6.1}$$

where v_j is the net internal activity level of neuron j, and y_j is the output of the neuron j. (2) The network contains one or more layers of hidden neurons that are not part of the input or output of the network. These hidden neurons enable the network to learn complex tasks by extracting progressively more meaningful features from the input patterns. In practice, we deploy more than one hidden layer. The first hidden layer acts as a detector of local features, and the second hidden layer acts as a detector of global features. Local features are extracted in the first hidden layer. Specifically, some neurons in the first hidden layer are used to partition the input space into regions, and other neurons in that layer learn the local features characterizing those regions. Global features are extracted in the second hidden layer. Specifically, a neuron in the second hidden layer combines the outputs of neurons in the first hidden layer operating on a particular region of the input space, and thereby learns the global features for that region. (3) The network exhibits a high degree of feed-forward connectivity, determined by the synapses of the network. It is through the combination of these characteristics together with the ability to learn from experience through training that the multilayer perceptron derives its computing power.

Multilayer perceptrons have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the back propagation algorithm. This algorithm is based on the error correction learning rule and will be discussed in the next subsection.

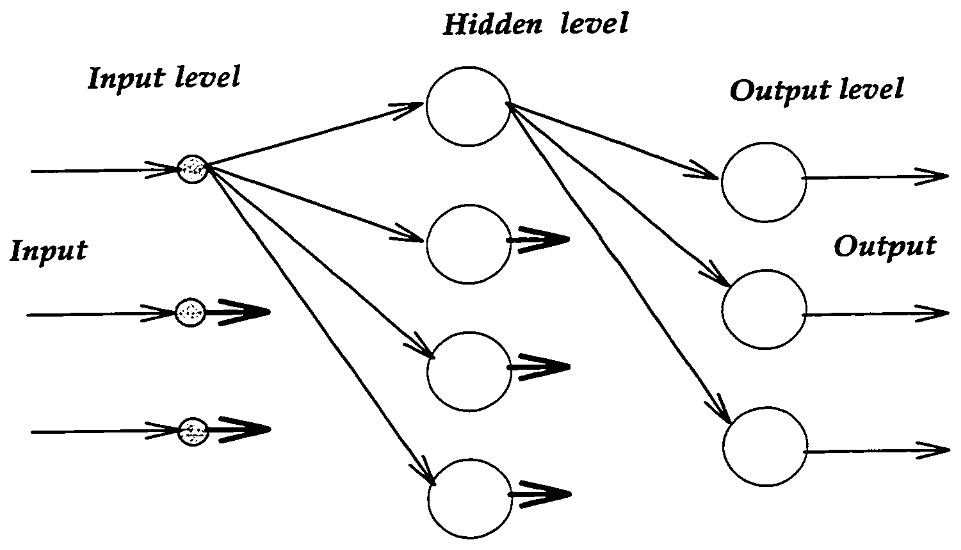


Figure 6.1: Multilayer perceptrons neural network.

6.2 Backpropagation Learning

6.2.1 Algorithm

The backpropagation process consists of two passes through the different layers of the network: a forward pass and a backward pass. During the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network, layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the network are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with the error-correction rule. The actual response of the networks is subtracted from a desired response to produce an error signal. This error signal is then propagated backward through the network, against the direction of synaptic connections. The synaptic weights are adjusted so as to make the actual response of the network move closer to the desired response. The mathematical development is described as follows [50, 89].

The error signal $e_j(n)$ at the output of neuron j at iteration n (i.e., presentation of the nth training pattern) is defined by

$$e_j(n) = d_j(n) - y_j(n), \tag{6.2}$$

where neuron j is an output node, $d_j(n)$ is the desired response of the jth neuron, and $y_j(n)$ is its output. We define the instantaneous value of the squared error for neuron j as $\frac{1}{2}e_j^2(n)$. Correspondingly, the instantaneous value $\zeta(n)$ of the sum of squared errors is obtained by summing $\frac{1}{2}e_j^2(n)$ over all neurons in the output layer; these are the only "visible" neurons for which error signals can be calculated. The instantaneous sum of squared errors of the network is thus written as

$$\zeta(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n), \tag{6.3}$$

where the set C includes all the neurons in the output layer of the network. Let N denote the total number of patterns contained in the training set. The average squared error is obtained by summing $\zeta(n)$ over all n and then normalizing with respect to the set size N, as shown by

$$\zeta_{av} = \frac{1}{N}\sum_{n=1}^{N} \zeta(n). \tag{6.4}$$

The instantaneous sum of error squares $\zeta(n)$, and therefore the average squared error ζ_{av}. is a function of all the free parameters (i.e., synaptic weights and thresholds) of the network. For a given training set, ζ_{av} represents the cost function as the measure of training set learning performance. The objective of the learning process is to adjust the free parameters of the network so as to minimize ζ_{av}. To do this minimization we use the technique that is very similar to LMS algorithm. Specifically, we consider a simple method of training in which the weights are updated on a pattern-by-pattern basis. The adjustments to the weights are made in accordance with the respective errors computed for each pattern presented to the network. The arithmetic average of these individual weight changes over the training set is therefore an estimate of the true change that would result from modifying the weights, based on minimizing the cost function ζ_{av} over the entire training set. In the multilayer perceptron network, the net internal activity level $v_j(n)$ produced at the input of the nonlinearity associated with neuron j is therefore

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n). \tag{6.5}$$

where p is the total number of inputs (including the threshold) applied to neuron j, $w_{ji}(n)$ denotes the synaptic weight connecting the output of neuron i to the input of neuron j. The synaptic weight w_{j0} (corresponding to the fixed input $y_0 = -1$) equals the threshold θ_j applied to neuron j. Hence the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n)). \tag{6.6}$$

In a manner similar to an LMS algorithm, the back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$, which is proportional to the instantaneous gradient $\partial\zeta(n)/\partial w_{ji}(n)$. According to the chain rule, we may express this gradient as follows:

$$\frac{\partial\zeta(n)}{\partial w_{ji}(n)} = \frac{\partial\zeta(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}\frac{\partial v_j(n)}{\partial w_{ji}(n)}. \tag{6.7}$$

The gradient $\partial\zeta(n)/\partial w_{ji}(n)$ represents a sensitivity factor, determining the direction of search in weight space for the synaptic weight $w_{ji}(n)$.

Differentiating both sides of Eq.6.3 with respect to $e_j(n)$, we get

$$\frac{\partial\zeta(n)}{\partial e_j(n)} = e_j(n). \tag{6.8}$$

Differentiating both sides of Eq.6.2 with respect to $y_j(n)$, we get

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1. \tag{6.9}$$

Next, differentiating Eq.6.6 with respect to $v_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)), \tag{6.10}$$

where the use of prime (on the right-hand side) signifies differentiation with respect to the argument. Finally, differentiating Eq.6.5 with respect to $w_{ji}(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \tag{6.11}$$

Hence, the use of Eqs.6.8 to 6.11 in Eq.6.7 yields

$$\frac{\partial\zeta(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n). \tag{6.12}$$

The correction $\triangle w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the *delta* rule

$$\triangle w_{ji}(n) = -\eta \frac{\partial \zeta(n)}{\partial w_{ji}(n)}, \tag{6.13}$$

where η is a constant that determines the rate of learning; it is called the learning-rate parameter of the back-propagation algorithm. The use of the minus sign in Eq.6.13 accounts for gradient descent in weight space. Accordingly, the use of Eq.6.12 in 6.13 yields

$$\triangle w_{ji}(n) = \eta \delta_j(n) y_i(n), \tag{6.14}$$

where the local gradient $\delta_j(n)$ is itself defined by

$$\delta_j(n) = -\frac{\partial \zeta(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi_j'(v_j(n)). \tag{6.15}$$

The local gradient points to required changes in synaptic weights. According to Eq.6.15, the local gradient $\delta_j(n)$ for output neuron j is equal to the product of the corresponding error signal $e_j(n)$ and the derivative $\varphi_j'(v_j(n))$ of the associated activation function.

From Eqs.6.14 and 6.15 we note that a key factor involved in the calculation of the weight adjustment $\triangle w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j. In this context, we may identify two distinct cases, depending on where in the network neuron j is located. In the case I, neuron j is an output node. This case is simple to handle, because each output node of the network is supplied with a desired response of its own, making it a straightforward matter to calculate the associated error signal. In case II, neuron j is a hidden node. Even though hidden neurons are not directly accessible, they share responsibility for any error made at the output of the network. The question is how to penalize or reward hidden neurons for their share of the responsibility. In the sequel, case I and case II are considered.

Case I: Neuron is an output node

When neuron j is located in the output layer of the network, it is supplied with a desired response of its own. Hence we may use Eq.6.2 to compute the error signal $e_j(n)$ associated with this neuron. Having determined $e_j(n)$, it is a straightforward matter to compute the local gradient $\delta_j(n)$ using Eq.6.15.

Case II: Neuron j is a hidden node

When neuron j is located in a hidden layer of the network, there is no specific desired response of its own. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected; this is where the development of the back-propagation algorithm gets complicated. According to Eq.6.15, we may redefine the local gradient $\delta_j(n)$ for hidden neuron j as

$$\delta_j(n) = -\frac{\partial\zeta(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial\zeta(n)}{\partial y_j(n)}\varphi_j'(v_j(n)). \tag{6.16}$$

where neuron j is hidden. To calculate the partial derivative $\frac{\partial\zeta(n)}{\partial y_j(n)}$, we use k instead of j in Eq.6.3 in order to avoid confusion with the use of the index j that refers to a hidden neuron under case II. In any event, differentiating Eq.6.3 with respect to the functional signal $y_j(n)$, we get

$$\frac{\partial\zeta(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)}. \tag{6.17}$$

Next, we use the chain rule for the partial derivative $\partial e_k(n)/\partial y_j(n)$, and rewrite Eq.6.17 in the equivalent form

$$\frac{\partial \zeta(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}. \tag{6.18}$$

However, from Eqs.6.2 and 6.6, we note that

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n)), \tag{6.19}$$

where neuron k is an output node. Hence,

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)). \tag{6.20}$$

We also note that for output neuron k, the net internal activity level is

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n), \tag{6.21}$$

where q is the total number of inputs (excluding the threshold) applied to neuron k. Here again, the synaptic weight $w_{k0}(n)$ is equal to the threshold $\theta_k(n)$ applied to neuron k, and the corresponding input y_0 is fixed at the value -1. In any event, differentiating Eq.6.21 with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n). \tag{6.22}$$

Thus, using Eqs.6.20 and 6.22 in 6.18, we get the desired partial derivative:

$$\frac{\partial \zeta(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k'(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n), \tag{6.23}$$

where we have used the definition of the local gradient $\delta_k(n)$ given in Eq.6.15 with the index k substituted for j.

Finally, using Eq.6.23 in 6.16, we get the local gradient $\delta_j(n)$ for hidden neuron j as follows:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n), \tag{6.24}$$

The factor $\varphi_j'(v_j(n))$ involved in the computation of the local gradient $\delta_j(n)$ in Eq.6.24 depends solely on the activation function associated with hidden neuron j. The remaining factor involved in this computation, namely, the summation over k, depends on two sets of terms. The first set of terms, the $\delta_k(n)$, requires knowledge of error signals $e_k(n)$, for all those neurons that lie in the layer to the immediate right of hidden neuron j, and that are directly connected to neuron j. The second set of terms, the $w_{kj}(n)$, consists of the synaptic weights associated with these connections.

We may now summarize the relations that we have derived for the back-propagation algorithm. First, the correction $\Delta w_{ji}(n)$ applied to the synaptic weight connecting to neuron i to neuron j is defined by the delta rule:

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning} \\ \text{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{input signal} \\ \text{of neuron } j \\ y_i(n) \end{pmatrix}. \tag{6.25}$$

Second, the local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:

1. If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and the error signal $e_j(n)$, both of which are associated with neuron j.

2. If neuron j is a hidden node, $\delta_j(n)$ equals the product of the associated derivative

$\varphi_j'(v_j(n))$ and the weighted sum of the $\delta's$ computed for the neurons in the next hidden or output layer that are connected to neuron j.

The two passes of computation

As mentioned at the beginning of this section, the back-propagation has two passes: a forward pass and a backward pass. In the forward pass the synaptic weights remain unaltered throughout the network, and the function signals of the network are computed on a neuron-by-neuron basis. Specifically, the function signal appearing at the output of neuron j is computed as

$$y_j(n) = \varphi(v_j(n)), \tag{6.26}$$

where $v_j(n)$ is the net internal activity level of neuron j, defined by

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n), \tag{6.27}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j, and $w_{ji}(n)$ is the synaptic weight connecting neuron i to neuron j , and $y_i(n)$ is the input signal of neuron j or, the function signal appearing at the output of neuron i. If neuron j is in the first hidden layer of the network, then the index i refers to the ith input terminal of the network, for which we write

$$y_i(n) = x_i(n), \tag{6.28}$$

where $x_i(n)$ is the ith element of the input vector. If, on the other hand, neuron j is in the output layer of the network, the index j refers to the jth output terminal of the network.

$$y_j(n) = o_j(n), \tag{6.29}$$

where $o_j(n)$ is the jth element of the output vector. This output is compared with the desired response $d_j(n)$, obtaining the error signal for the jth output neuron. Thus, the forward phase of computation begins at the first hidden layer by presenting it with the input vector, and terminates at the output layer by computing the error signal for each neuron of this layer.

The backward pass starts at the output layer by passing the error signals backward through the network, layer by layer, and recursively computing the δ for each neuron. This recursive process permits the synaptic weights of the network to undergo changes in accordance with the delta rule of Eq.6.25. For the neuron located in the output layer, the δ is simply equal to the error signal of that neuron multiplied by the first derivative of its nonlinearity. Hence, we use Eq.6.25 to compute the changes to the weights of all the connections feeding into the output layer. Given the $\delta's$ for the neurons of the output layer, we can use Eq.6.24 to compute the $\delta's$ for all the neurons in the penultimate layer and therefore the changes to the weights of all connections feeding into it. The recursive computation is continued, layer by layer, by propagating the changes to all synaptic weights. In each forward and backward propagating process, the input pattern is fixed throughout the round-trip phase.

The local gradient of sigmoidal neuron

The computation of the δ for each neuron of the multilayer perceptron requires an explicit formation of the derivative of the activation function $\varphi(.)$ associated with that neuron. For this derivative to exist, we require the function $\varphi(.)$ to be continuous. In basic terms, differentiability is the only requirement that an activation function would have to satisfy. In the multilayer perceptron networks which we will use in the later section, the sigmoidal nonlinear activation function is used, a particular form is the logistic function:

$$y_j(n) = \varphi_j(v_j(n)) = \frac{1}{1+\exp(-v_j(n))}, \quad -\infty < v_j(n) < \infty, \tag{6.30}$$

where $v_j(n)$ is the net internal activity level of neuron j. According to this nonlinearity, the amplitude of the output lies inside the range $0 \leq y_j \leq 1$. The δ for the multilayer perceptron network with each neuron having the logistic function is generated as follows. Differentiating both sides of Eq.6.30 with respect to $v_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) = \frac{\exp(-v_j(n))}{[1+\exp(-v_j(n))]^2}. \tag{6.31}$$

Using Eq.6.30 to eliminate the exponential term $\exp(-v_j(n))$ from Eq.6.31, we may express the derivative of activation function as

$$\varphi_j'(v_j(n)) = y_j(n)[1 - y_j(n)]. \tag{6.32}$$

For a neuron j located in the output layer, we note that $y_j(n) = o_j(n)$. Hence, we may express the local gradient for neuron j as

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) = [d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)], \tag{6.33}$$

where $o_j(n)$ is the function signal at the output of neuron j, and $d_j(n)$ is the desired response for it. On the other hand, for an arbitrary hidden neuron j, we may express the local gradient for neuron j as

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) = y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n). \qquad (6.34)$$

It can be seen that from Eq.6.32 the derivative $\varphi_j'(v_j(n))$ attains its maximum value at $y_j(n) = 0.5$, and its minimum value at $y_j(n) = 0$, or $y_j(n) = 1.0$. Since the amount of change in a synaptic weight of the network is proportional to the derivative $\varphi_j'(v_j(n))$, it follows that for a logistic activation function the synaptic weights are changed the most for those neurons in the network for which the function signals are in their midrange. This phenomenon will guide us to initialize neurons and change the rate of learning.

Rate of learning

The back-propagation algorithm provides an "approximation" to the trajectory in the weight space computed by the method of steepest descent. The smaller we make the learning-rate parameter η, the smaller will the changes to the synaptic weights in the network be from one iteration to the next and the smoother will be the trajectory in the weight space. This improvement is attained at the cost of a slower rate of learning. If, on the other hand, we make the learning-rate parameter η too large so as to speed up the rate of learning, the resulting large changes in the synaptic weights assume such a form that the network may become unstable (i.e., oscillatory). A simple method of increasing the rate of learning and yet avoiding the danger of instability is to modify the delta rule of Eq.6.14 by including a momentum term as follows:

$$\triangle w_{ji}(n) = \alpha \triangle w_{ji}(n-1) + \eta \delta_j(n) y_i(n), \tag{6.35}$$

where α $(0 \leq |\alpha| < 1)$ is usually a positive number called the momentum constant. Eq.6.35 is called the generalized delta rule; it includes the delta rule of Eq.6.14 as a special case.

The incorporation of momentum in the back-propagation algorithm represents a minor modification to the weight update, and yet it can have highly beneficial effects on the learning behavior of the algorithm. The momentum term may also have the benefit of preventing the learning process from terminating in a shallow local minimum on the error surface. We will say more on learning rate related to the classification network training in the next section.

Pattern and batch modes of training

In the practical application of the back-propagation algorithm. learning results from the many presentations of a prescribed set of training examples to the multilayer perceptron. One complete presentation of the entire training set during the learning process is called an epoch. The learning process is maintained on an epoch-by-epoch basis until the synaptic weights and threshold levels of the network stabilize and the average squared error over the entire training set converges to some minimum value. It is good practice randomizing the order of presentation of training examples from one epoch to the next. This randomization tends to make the search in the weight space stochastic over the learning cycles, thus, avoiding the possibility of limit cycles in the evolution of the synaptic weight vector. For a given training set, back-propagation learning may proceed in one of two basic ways:

Pattern mode. In the pattern mode of back-propagation learning, weight updating is performed after the presentation of each training example. To be specific, consider an epoch consisting of N training examples arranged in the order $[x(1), d(1)], ..., [x(N), d(N)]$. The first example $[x(1), d(1)]$ in the epoch is presented to the network, and the sequence of forward and backward computations described previously is performed, resulting in certain adjustments to the synaptic weights and threshold levels of the network. Then the second example $[x(2), d(2)]$ in the epoch is presented, and the sequence of forward and backward computations is repeated, resulting in further adjustments to the synaptic weights and threshold levels. This process is continued until the last example $[x(N), d(N)]$ in the epoch is accounted for.

Batch mode. In the batch mode of back-propagation learning, weight updating is performed after the presentation of all the training examples that constitute an epoch. For a particular epoch, we define the cost function as the average squared error given by Eqs.6.3 and 6.4, reproduced here in the composite form:

$$\zeta_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n), \tag{6.36}$$

where the error signal $e_j(n)$ pertains to output neuron j for training example n and which is defined by Eq.6.2. The error $e_j(n)$ equals the difference between $d_j(n)$ and $y_j(n)$, which represent the jth element of the desired response vector $\mathbf{d}(n)$ and the corresponding value of the network output, respectively. In Eq.6.36, the inner summation with respect to j is performed over all the neurons in the output layer of the network, whereas the outer summation with respect to n is performed over the entire training set in the epoch. For a learning-rate parameter η, the adjustment applied to synaptic weight w_{ji}, connecting neuron i to neuron j, is defined by the delta rule

$$\triangle w_{ji} = -\eta \frac{\partial \zeta_{av}}{\partial w_{ji}(n)} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}}. \qquad (6.37)$$

To calculate the partial derivative $\partial e_j(n)/\partial w_{ji}$ we proceed in the same way as before. According to Eq.6.37, in the batch model the weight adjustment $\triangle w_{ji}$ is made only after the entire training set has been presented to the network.

The pattern mode of training is preferred over the batch mode for "on-line" operation, because it requires less local storage for each synaptic connection. Moreover, given that the patterns are presented to the network in a random manner, the use of pattern-by-pattern updating of weights makes the search in weight space stochastic in nature, which, in turn, makes it less likely for the back-propagation algorithm to be trapped in a local minimum. On the other hand, the use of the batch mode of training provides a more accurate estimate of the gradient vector. The mode which is used in practice is determined by the problem at hand.

Summary

Suppose $[\mathbf{x(n)}, \mathbf{d(n)}]$ is a training pair, namely, $\mathbf{x(n)}$ is the input pattern vector and $\mathbf{d(n)}$ is the desired output vector. The pattern-by-pattern updating of weights in backpropagation algorithm for L layer neurons cycles through the training data

$\{[\mathbf{x(n)}, \mathbf{d(n)}];\ n = 1, 2, \ldots N\}$ as follows:

1. Initialization. Start with a reasonable network configuration. and set all the synaptic weights and threshold levels of the network to small random numbers that are uniformly distributed.

2. Presentations of Training Examples. Present the network with an epoch of training examples. for each example. perform the following sequence of forward and backward computation.

3. Forward Computation. Let a training example in the epoch be denoted by $[\mathbf{x(n)}, \mathbf{d(n)}]$, with the input vector $\mathbf{x(n)}$ applied to the input layer of sensory nodes and the desired response vector $\mathbf{d(n)}$ presented to the output layer of computation nodes. Compute the activation potentials and function signals of the network by proceeding forward through the network, layer by layer. The net internal activity level $v_j^l(n)$ for neuron j in layer l is

$$v_j^l(n) = \sum_{i=0}^{p} w_{ji}^l(n) y_i^{l-1}(n). \tag{6.38}$$

where p is the number of synaptic weights connecting to neuron j, $y_i^{l-1}(n)$ is the function signal of neuron i in the previous layer $l-1$ at iteration n and $w_{ji}^l(n)$ is the synaptic weight of neuron j in layer l that is fed from neuron i in layer $l-1$. For $i = 0$, we have $y_0^{l-1}(n) = -1$ and $w_{j0}^l(n) = \theta_j^l(n)$, where $\theta_j^l(n)$ is the threshold applied to neuron j in layer l. Assuming the use of a logistic function in neuron response, the function signal of neuron j in layer l is

$$y_j^l(n) = \frac{1}{1+\exp(-v_j^l(n))}. \tag{6.39}$$

If neuron j is in the first hidden layer (i.e., $l = 1$), set

$$y_j^0(n) = x_j(n), \tag{6.40}$$

where $x_j(n)$ is the jth element of the input vector $\mathbf{x(n)}$. If neuron j is in the output layer (i.e. $l = L$), set

$$y_j^L(n) = o_j(n), \tag{6.41}$$

where $o_j(n)$ is the output of jth neuron in output layer. Hence, compute the error signal

$$e_j(n) = d_j(n) - o_j(n), \tag{6.42}$$

where $d_j(n)$ is the jth element of the desired response vector $\mathbf{d(n)}$.

4. Backward Propagation. Compute the local gradients δ of the network by proceeding backward, layer by layer:

$$\delta_j^L(n) = e_j^L(n)o_j(n)[1-o_j(n)] \quad \text{for neuron } j \text{ in output layer.} \tag{6.43}$$

$$\delta_j^l(n) = y_j^l(n)[1-y_j^l(n)]\sum_k \delta_k^{l+1}(n)w_{kj}^{l+1}(n) \quad \text{for neuron } j \text{ in hidden layer } l. \tag{6.44}$$

Hence, we adjust the synaptic weight of the network in layer l according to the generalized delta rule:

$$w_{ji}^l(n+1) = w_{ji}^l(n) + \alpha[w_{ji}^l(n) - w_{ji}^l(n-1)] + \eta\delta_j^l(n)y_i^{l-1}(n), \tag{6.45}$$

where η is the learning rate parameter and α is the momentum constant.

5. Iteration. Iterate the computation by presenting new epochs of training examples to the network until the free parameters of the network stabilize their values and the average squared error computed over the entire training set is at a minimum or acceptably small value.

The above process which minimizes the average squared error is tantamount to maximizing an *a posteriori* probability function $E[\mathbf{d(n)}|\mathbf{x(n)}]$ for every n, in which E is the expectation operator.

6.2.2 Initialization

The first step in back-propagation learning is to initialize the network. A good choice for the initial values of the free parameters of the network can be of tremendous help in a successful network design. In cases where prior information is available, it may be better to use the prior information to guess the initial values of the free parameter. But how do we initialize the network if no prior information is available? The customary practice is to set all the free parameters of the network to random numbers that are uniformly distributed inside a small range of values.

The wrong choice of initial weights can lead to a phenomenon known as premature saturation. This phenomenon refers to a situation where the instantaneous sum of squared error $\zeta(n)$ remains almost constant for some period of time during the learning process. Such a phenomenon can not be considered as a local minimum, because the squared error continues to decrease after this period is finished.

When a train pattern is applied to the input layer of a multilayer perceptron, the output values of the network are calculated through a sequence of forward computations that involves inner products and sigmoidal transformations. This is followed by a sequence of backward computations that involves the calculation of error signals and pertinent slope of the sigmoidal activation function to form a synaptic weight adjustment. Suppose that, for a particular pattern, the net internal activity level of an output neuron is computed to have a large magnitude. Then assuming that the sigmoidal activation function of the neuron has the limiting values 0 and 1, we find that the corresponding slope of the activation function for that neuron will be very small, and the output value for the neuron will be close to 0 or 1. In such a case, we say that the neuron is in "saturation". If the output value is close to 0 when the target value is 1, or vice versa, we say that the neuron is incorrectly "saturated". When this happens, the adjustment applied to the synaptic weights of the neuron will be small, and the network may take a long time to escape it.

At the initial stage of back-propagation learning, both unsaturated neurons and incorrectly saturated ones may exist in the output layer of the network. As the learning process continues, the synaptic weights associated with unsaturated output neurons change rapidly, because the corresponding error signals and gradients have relatively large magnitudes, thereby resulting in a reduction in the instantaneous sum of squared error. If, however, at this point the incorrectly saturated output neurons remain saturated for some particular training patterns, then the phenomenon of premature saturation may arise. The phenomenon can be avoided by choosing the initial values of the synaptic weights and threshold levels of the network to be uniformly distributed inside a small range of values.

6.2.3 Stopping Criteria

There are some reasonable criteria each with its own practical merits, which may be used to terminate the weight adjustments. To formulate such a criterion, the logical thing to do is to think in terms of the unique properties of a local or global minimum of the error surface. Let us denote an optimal weight vector $\mathbf{w}^*$ which corresponds to a local or global minimum of the error surface. A necessary condition for $\mathbf{w}^*$ to be optimal is that the gradient vector $\partial\zeta(n)/\partial\mathbf{w}$ of the error surface be zero at $\mathbf{w} = \mathbf{w}^*$. Accodingly, we may formulate a sensible convergence criterion for back-propagation learning:

The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

The drawback of this convergence criterion is that, for successful trials, the learning time may be long. Also, it requires the computation of the gradient vector. Another unique property of a minimum is the fact that the cost function or error measure is stationary at the point $\mathbf{w} = \mathbf{w}^*$. We may suggest a different criterion of convergence:

The back-propagation algorithm is considered to have converged when the *absolute rate of change in the average squared error per epoch is sufficiently small.*

Typically, the rate of change in the average squared error is considered to be small enough if it lies in the range of 0.1 to 1 percent per epoch. The threshold depends on the problem at hand.

6.3 Target Classification

In order to classify targets, it is necessary to identify the target features. In the simulations presented here, a feature set, which possesses the dominant characteristics of targets and has a certain feature dynamic range, is chosen (see target characteristics). The entire classification system consists of identification nets and classification net. Each identification net is used to extract a particular feature of the targets, the outputs of which are fused by a classification net. Each net is trained by training set with BP to achieve a maximal *a posteriori* probability $E[\mathbf{d}(\mathbf{n})|\mathbf{x}(\mathbf{n})]$ whereby the target feature information has been stored in the neural network "memory". For the positive logic, when the test datum $\mathbf{x}$ is presented to the network, the output of network stands for approximately an *a posteriori* probability $E[\mathbf{d}|\mathbf{x}]$. Given a threshold, we can make a decision that the target feature is detected or not, or which cluster the target belongs to. In some applications, neuron firing state stands for one feature and off another.

6.3.1 Target Characteristics

In the simulation, three flight target features are deployed: radar cross-section (RCS) of targets, Doppler frequencies and target traces. Therefore three corresponding sensors are used to obtain three messages quantitatively. MTI radar works at X-band (9.83 GHz). Suppose that there are two targets. The first target has a radar cross-section range $[0.8 - 1.3]m^2$, Doppler frequency range $[200 - 1200]$ Hz, the primary Doppler frequency caused by the flight body has a range $[245 - 308]$ Hz, and flight route: $y = 1.3x + 3 + n$, where n is measurement error. The second target has a radar cross-section range $[2.7 - 3.2]m^2$, Doppler frequency range $[200 - 1200]Hz$, the primary Doppler frequency caused by the flight body has a range $[400 - 460]$ Hz, and flight route: $y = 0.8x + 2 + n$. This is the situation we simulated.

6.3.2 Architecture

The design layout of neural networks is shown in Fig.6.2. There are three identification (ID) nets and one classification net (CN). In the first ID net, the input layer has 32 source nodes which will sense the target Doppler frequency. The hidden layer has 24 neurons. The output layer has one neuron. Its output "1" stands for the first target, "0" for the second target. In the second ID net, the input layer has 2 source nodes, which will perceive flight route data. The hidden layer has 16 neurons. The output layer has 2 neurons. The output (1,0) stands for the first target, (0,1) for the second target. In the third ID net, the input layer has one source node which will feel the

RCS. The hidden layer has 16 neurons and the output layer has two neurons. The output (0,1) stands for the first target, and (1,0) for the second target. In the CN, the input layer has five source nodes which will collect data from ID nets. The hidden layer has 10 neurons and the output layer has 3 neurons. The output (1,0,0) stands for the first target, (0,1,0) for the second target, and (0,0,1) for unknown targets. The number of neurons in the hidden layer of each net is determined by trial to ensure the convergence in training process.

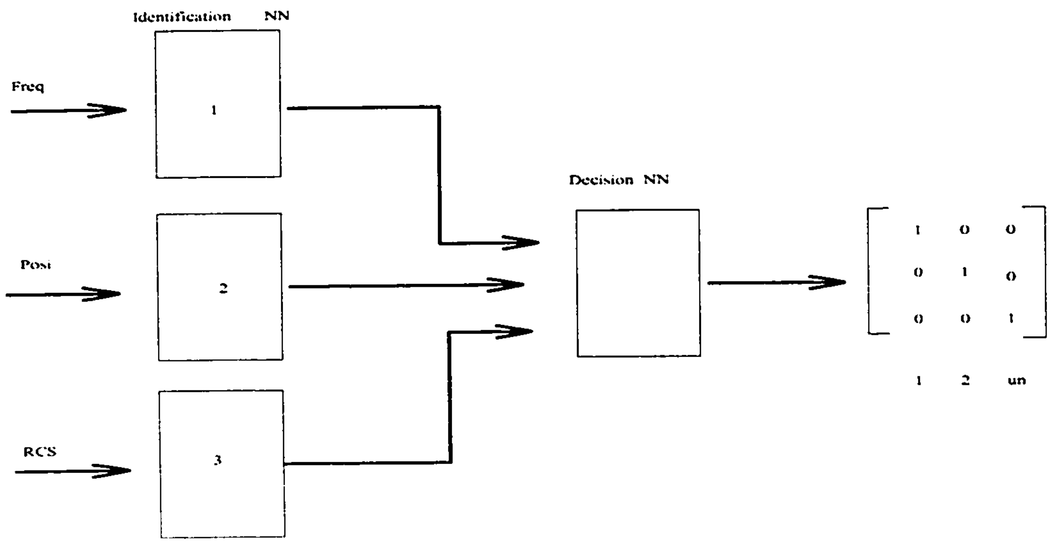


Figure 6.2: Layout of classification networks.

6.3.3 Training and Testing

The target echo (received signal) is estimated by an AR model frequency estimator. The spectrum from the AR model frequency estimator at 32 discrete points is fed to the source nodes of the first ID net. Herein, 60 temporal intervals are used. 32-by-60 batch input matrix is formed and 1-by-60 batch target vector corresponding to the batch input matrix is formed in accordance with different target echo. By analogy, we can form batch input and batch target for the second ID net, the third ID net and CN.

In Fig.6.3 is shown one sample of the first target's Doppler frequencies, which is the output of the AR spectrum estimator. In Fig.6.4 is shown one sample of the second target's Doppler frequencies. In Fig.6.5 are given the RCS distributions which are used to train the RCS ID network. In Fig.6.6 are presented the target tracks which are used to train the position network.

A multilayer perceptrons trained with the back-propagation algorithm may learn faster (in terms of the number of training iterations required) when the sigmoidal activation function built into the neuron model of the network is asymmetric than when it is nonsymmetric. We say that an activation function $\varphi(v)$ is asymmetric if $\varphi(-v) = -\varphi(v)$. This condition is not satisfied by the logistic function. Based on the stipulation of the architecture for the neural target classifier, we use the logistic function for the neuron model in the output layer; in the first hidden layer, the asymmetric activation function – hyperbolic tangent, is deployed for the neuron model

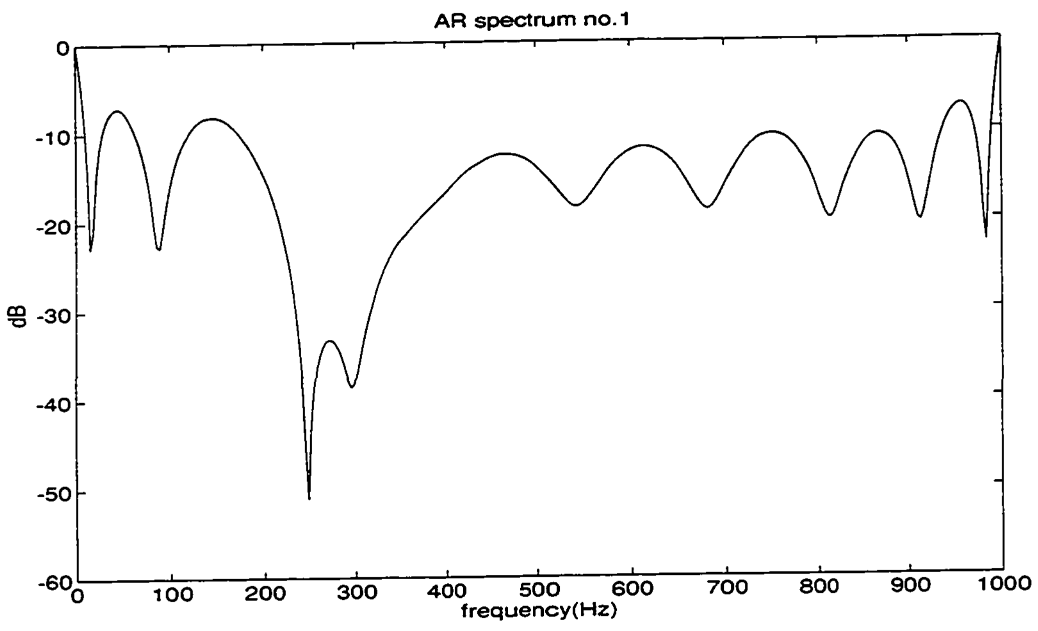


Figure 6.3: One sample of the first target's Doppler frequency for training the first ID net.

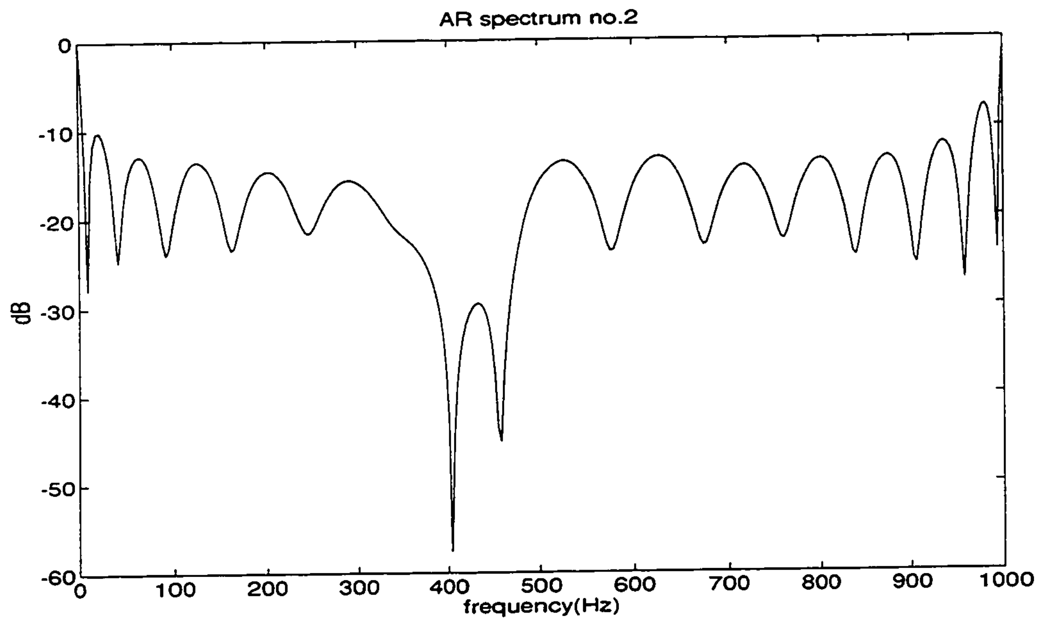


Figure 6.4: One sample of the second target's Doppler frequency for training the first ID net.

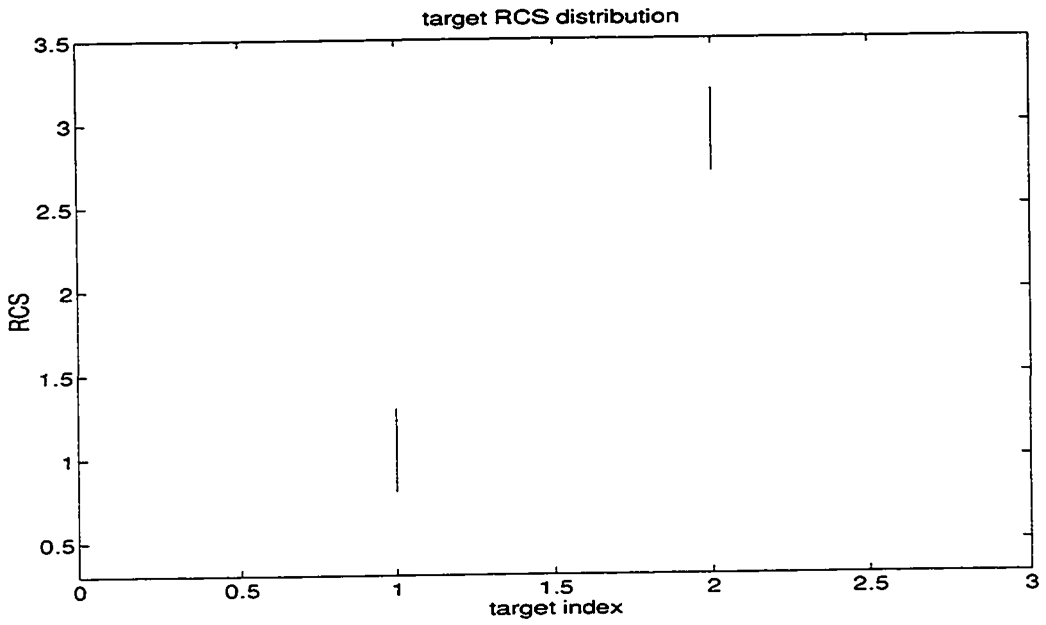


Figure 6.5: RCS distribution of target 1 and target 2 for training the RCS ID net.

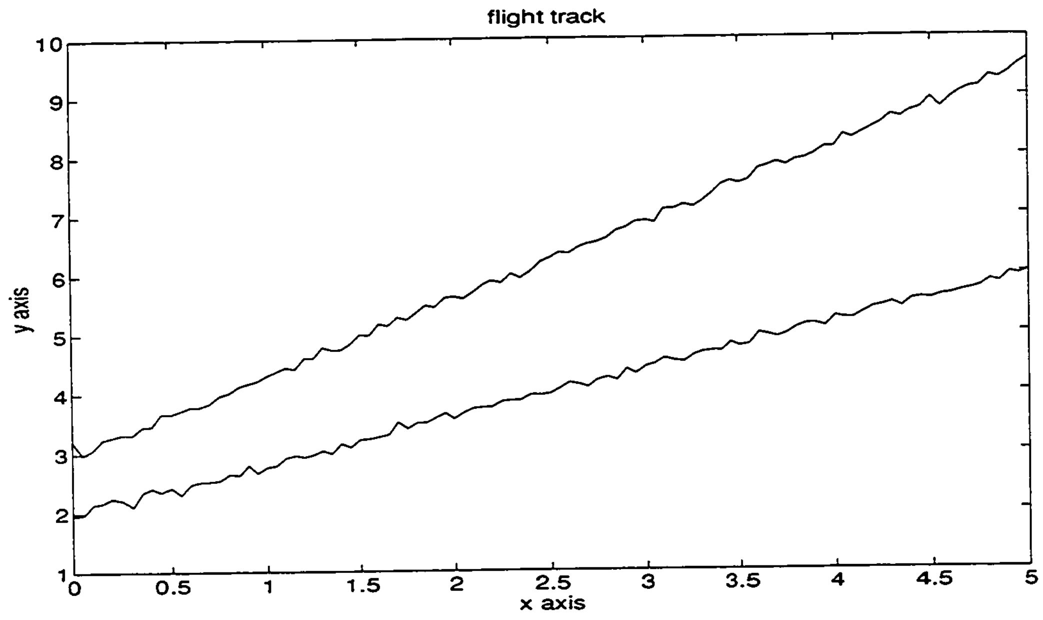


Figure 6.6: Routes of target 1 and target 2 for training the position ID net.

to speed up mapping process.

The batch mode is used in the training network: the weight updating is performed after the presentation of all the training examples that constitute an epoch, instead of updating pattern by pattern. In our experiment, both momentum and adaptive learning rate are deployed to improve the training performance. The momentum is set to be 0.95. The learning rate changes as follows:

If the new error exceeds the old error by more than a predefined ratio (typically 1.04), the new weights, biases and output error are discarded. In addition, the learning rate is decreased (typically by multiplying by 0.7). Otherwise the new weights, etc. are kept. If the new error is less than the old error, the learning rate is increased (typically by multiplying by 1.05).

In Fig.6.7 is given the learning curve (the summation of squared errors is plotted against the training epoch) for the frequency ID net. In Fig.6.8 is shown the learning curve of the RCS recognition network. In Fig. 6.9 is presented the learning curve for the CN.

Testing is carried out to estimate the classification performance of the neural network. Both training data and interpolating data are tested. When we set the detection threshold at 0.96 in CN, the classification capacity is 100%.

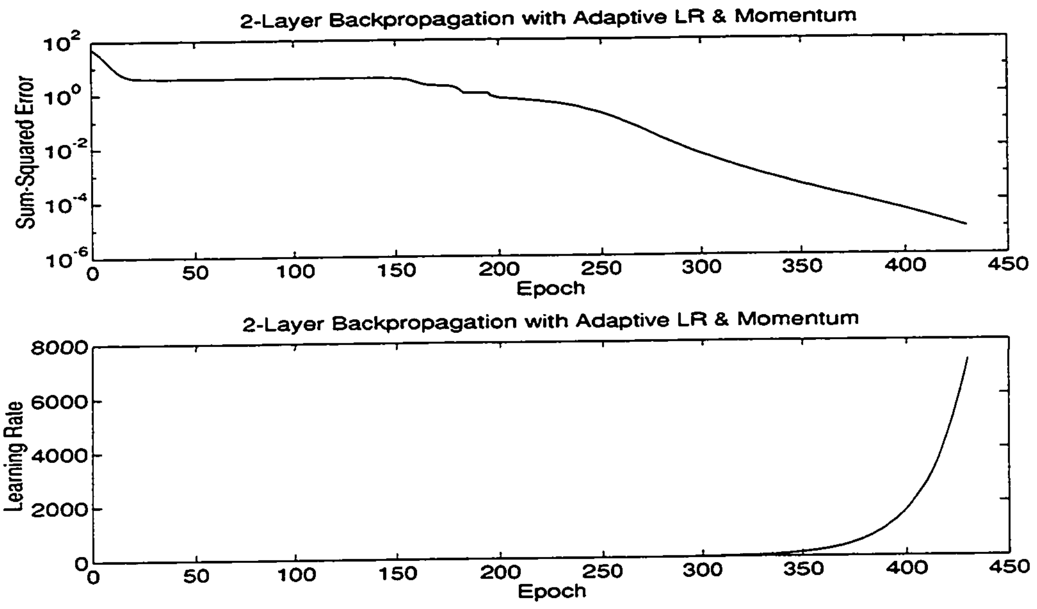


Figure 6.7: Learning curve of frequency ID net.

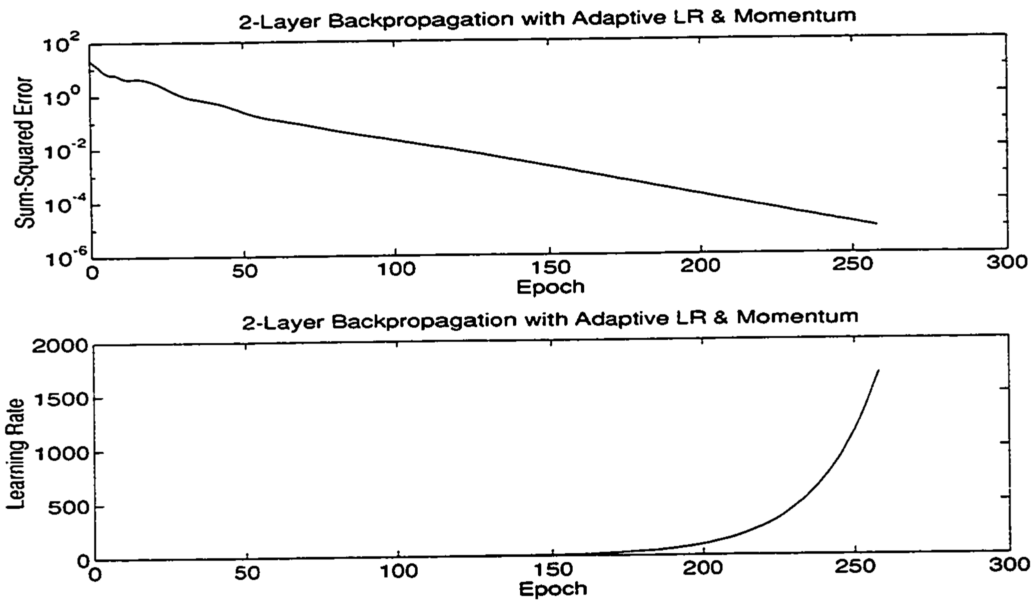


Figure 6.8: Learning curve of RCS ID network.

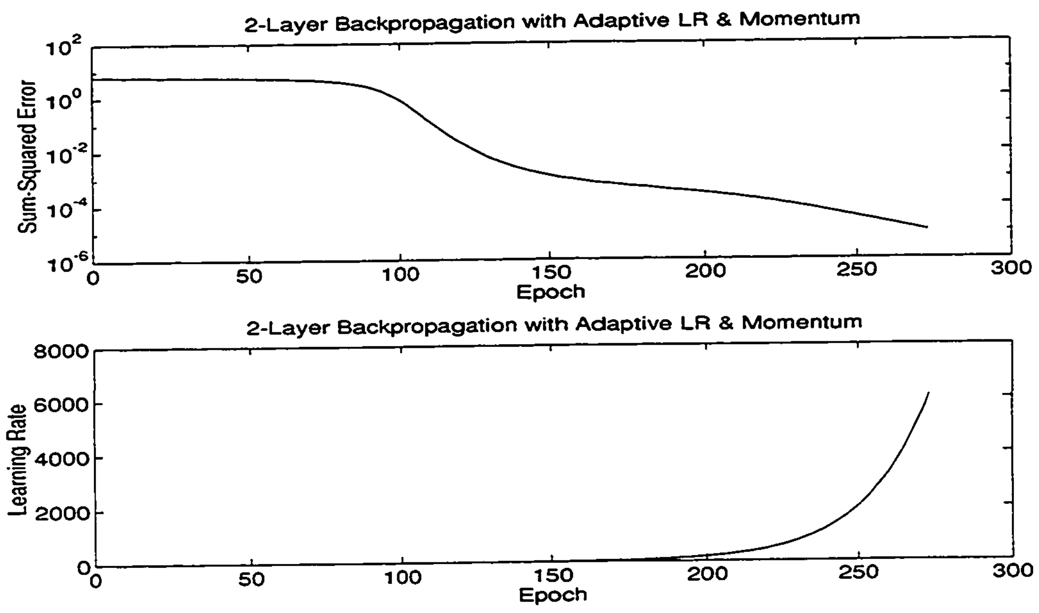


Figure 6.9: Learning curve of classification network.

6.4 Discussions of Target Classification

Optimum data fusion in CN is performed with multilayer perceptrons trained by BP. Alternatively, it can be implemented by a linear combination of ID net decision outputs followed by the application of a threshold. The optimum weight vector for the linear combination is a function of the performance probabilities of ID nets. This fusion algorithm is equivalent to a perceptron for which the weight vector can be adapted by the error correction rule. The perceptron is the simplest form of a neural network used for the classification of a special type of patterns said to be linearly separable. It consists of a single neuron with adjustable synaptic weights and threshold. The weights can be adapted using an error correction rule given by [50].

Neural networks for data fusion have proven to be a flexible solution. It is easy to extend them to accommodate future changes. More importantly, without modifying the general structures of the networks, additional training facts can be added to the training sets so as to include more target features or perform more complex tasks. If more signatures will be used to carry out the target classification, we can use additional identification networks and increase the number of input nodes of the classification networks. If more than two targets are to be classified, the functions of classification network can be extended by increasing the number of output neurons. In addition, once neural networks are trained well, real-time data fusion can be realized since neural network operation is parallel, which makes neural networks have prominent advantages over other devices.

It is worthwhile noting that in the presence of noise, the features contaminated by the noise should be used to train the networks. When the features overlap, more hidden layers should be used to increase the generalization ability of the networks. In this case, the training speed will be slow.

The real data performance test for multilayer perceptron networks should be carried out for further research.

Chapter 7

Feature Mapping Data Fusion

7.1 Introduction

The problem of multitarget tracking using multiple sensors has received considerable attention in recent years. The pay-off that results when one combines data from a variety of sensors is obvious. Basically, it consists of an enhancement to the over all system performance. Sensor data fusion may be carried out by either combining sensor tracks or sensor observations. Track fusion was studied by Singer and Kanyuck [63], who assumed independent noise process at each sensor. A practical implementation is given in [64]. Bar-Shalom refined the track fusion equations by accounting for the common process noise that is observed at the sensors [65]. An alternative approach is to fuse sensor observations rather than tracks [52] [66]. In [66], Roecker and McGillem compared tracking fusion and measurement fusion methods, and gave an example which showed the improvement in the uncertainty of the resulting estimate of the state vector when the measurement fusion method was used. In this chapter we will discuss a new approach to the measurement data fusion.

The measurement fusion (plot-level) tracking model is shown in Fig.7.1. In this model, the sensors produce streams of plots or individual target detections which are processed by a single tracking filter to generate the integrated database directly.

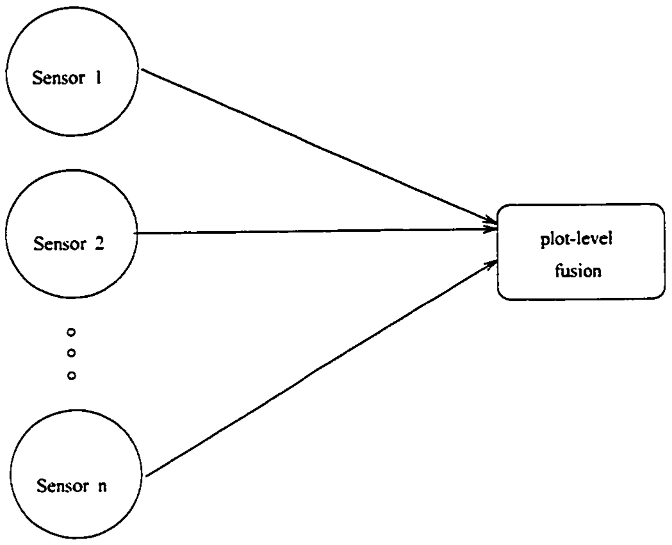


Figure 7.1: Centralized plot-level data fusion.

This approach is recognized as having the advantages of (i) generating accurate continuous tracks, (ii) being free of error correlation problems, (iii) having a low total computation load, (iv) being more amenable to sophisticated techniques. It has the disadvantage of being susceptible to sensor degradation. In the case of the plot-level data fusion, data clustering has to be carried out before conventional methods are applied. This turns out to be a very difficult problem to solve when targets are close to one another. In this chapter, a centralized plot-level data fusion technique, which is based on a neural network, is presented. A self-organized feature-mapping technique, which learns from sensor observations, is used to integrate the data from several sensors with various unknown measurement accuracies. This new technique for data fusion is termed as feature mapping data fusion (FMDF). Topological neighbourhood formulation among the network for integrating the data is described. Since the neural network learns from the sensor observations. it is not necessary to cluster the data. In the development of this approach. it is supposed that coordinate transformation and synchronization have been done before the observed data are transferred to a fusion processor. Through computer simulations, the feature-mapping fusion ability is studied. Data fusion results are presented herein. As well, comparisons are made between the results obtained using the neural network method. Maximum Likelihood and other *ad hoc* methods.

7.2 Feature-Mapping Learning

Self-organized feature mapping is based on *competitive learning*. The output neurons of the network compete among themselves to be activated or fired. with the result that only one output neuron. or one neuron per group. is on at any one time. The output neurons that win the competition are called "winner-takes-all" neurons. One way of inducing a "winner-takes-all" competition among the output neurons is to use lateral inhibitory connections (i.e.. negative feedback paths) between them.

In a self-organized feature map, the neurons are placed at the nodes of a lattice that is usually one- or two-dimensional; higher dimensional maps are also possible but not common. The neurons become selectively tuned to various input patterns or classes of input patterns in the course of a competitive learning process. The locations of the winner neurons tend to become ordered with respect to each other in such a way that a meaningful coordinate system for different input features is created over the lattice [67]. A self-organizing feature map is therefore characterized by the formulation of a topographic map of the input patterns, in which the spatial locations (i.e., coordinates) of the neurons in the lattice correspond to intrinsic features of the input patterns, hence the name "self-organizing feature mapping."

The development of this special class of artificial neural network is motivated by a distinct feature of the human brain [50]; the brain is organized in many places in such a way that different sensory inputs are represented by topologically ordered computation maps. In particular, sensory inputs such as tactile, visual, and acoustic are mapped onto different areas of the cerebral cortex in a topologically ordered manner. Thus the computational map constitutes a basic building block in the information-processing infrastructure of the nervous system. A computational map is defined by an array of neurons representing slightly differently tuned processors or filters, which operate on the sensory information-bearing signals in parallel. Consequently, the neurons transform input signals into a place-coded probability distribution that represents the computed values of parameters by sites of maximum relative activity within the map. The information so derived is of such a form that it can be readily accessed by higher-order processors using relatively simple connections.

7.2.1 Basic Feature-Mapping Models

Kohonen [67] found that the spatial location of an output neuron in the topographic map corresponds to a particular domain or feature of the input data. The output neurons are usually arranged in a one- or two-dimensional lattices, a topology that ensures that each neuron has a set of neighbors.

The manner in which the input patterns are specified determines the nature of the feature-mapping model. In particular, we may distinguish two basic models, as illustrated in Figs.7.2 and 7.3 for a two-dimensional lattice of output neurons that are fully connected to the inputs. Both models were inspired by the pioneering self-organizing studies of von der Malsburg, who noted that a model of the visual cortex could not be entirely genetically predetermined; rather, a self-organizing process involving synaptic learning may be responsible for the local ordering of feature-sensitive cortical cells. However, global topographic ordering was not achieved, because the model used a fixed neighborhood.

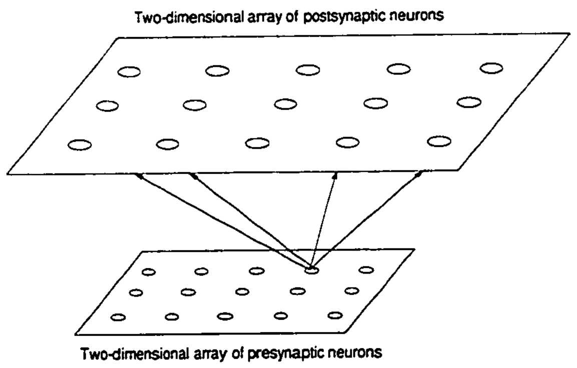

Two-dimensional array of postsynaptic neurons
Two-dimensional array of presynaptic neurons

Figure 7.2: Willshaw-von der Malsburg's model.

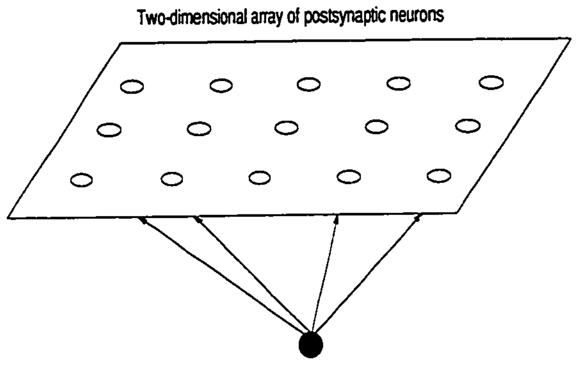


Figure 7.3: Kohonen's model.

The model of Fig.7.2 was originally proposed by Willshaw and von der Malsburg on biological grounds to explain the problem of retinotopic mapping from the retina to the visual cortex. Specifically, there are two separate two-dimensional lattices of neurons connected together. One lattice represents presynaptic (input) neurons and the other lattice represents postsynaptic (output) neurons. The postsynaptic lattice uses a short-range excitatory mechanism as well as a long-range inhibitory mechanism. These two mechanisms are local in nature and critically important for self-organization. The two lattices are interconnected by modifiable synapses of a Hebbian type. Strictly speaking, therefore, the postsynaptic neurons are not winner-takes-all; rather, a threshold is used to ensure that only a few postsynaptic neurons will fire at any one time. Moreover, to prevent a steady buildup in the synaptic weights that may lead to network instability, the total weight associated with each postsynaptic neuron is limited by an upper boundary condition. Thus, for each neuron, some synaptic weights increase while others are made to decrease. The basic idea of the Willshaw-van der Malsburg model is for the geometric proximity of presynaptic neurons to be coded in the form of correlations in their electrical activity, and to use these correlations in the postsynaptic lattice so as to connect neighboring presynaptic neurons to neighboring postsynaptic neurons. A topologically ordered mapping is therefore produced by self-organization. Note , however, that the Willshaw-von der Malsburg model is specialized to mappings where the input dimension is the same as the output dimension.

The second model, of Fig.7.3, introduced by Kohonen, is not meant to explain neurobiological details. Rather, the model tries to capture the essential features of computational maps in the brain and yet remain computationally tractable. The model's neurobiological feasibility is discussed by Kohonen. It appears that the Kohonen model is more general than the Willshaw-von der Malsburg model in the sense that it is capable of performing data compression.

In reality, the Kohonen model belongs to the class of vector coding algorithm. We say so because the model provides a topological mapping that optimally places a fixed number of vectors into a higher-dimensional input space, and thereby facilitates data compression. The Kohonen model may therefore be derived in two ways. We may use basic ideas of self-organization, motivated by neurobiological considerations, to derive the model, which is the traditional approach. Alternatively, we may use a vector quantization approach that uses a model involving an encoder and a decoder, which is motivated by communication-theoretic considerations. The derivation of the self-organizing feature-mapping (SOFM) usually associates with the Kohonen model, its basic properties, and applications.

7.2.2 Function of Lateral Feedback

In order to pave the way for the development of self-organizing feature maps, we first discuss the use of lateral feedback as a mechanism for modifying the form of excitation applied to a neural network. By lateral feedback we mean a special form of feedback that is dependent on lateral distance from the point of its application.

For the purpose of this discussion, it is adequate to consider the one-dimensional lattice of neurons shown in Fig.7.4, which contains two different types of connections. These are forward connections from the primary source of excitation, and those that are internal to the network by virtue of self-feedback and lateral feedback. In Fig.7.4, the input signals are applied in parallel to the neurons. These two types of local connections serve two different purposes. The weighted sum of the input signals at each neuron is designed to perform feature detection. Hence each neuron produces a selective response to a particular set of input signals. The feedback connections, on the other hand, produce excitatory or inhibitory effects, depending on the distance from neuron.

The neural network described here exhibits two important characteristics due to the lateral feedbacks. First, the network tends to concentrate its electrical activity into local clusters, referred to as activity bubbles. Second, the locations of the activity bubbles are determined by the nature of the input signals.

Let $x_1, x_2, ..., x_p$ denote the input signals applied to the network, where p is the number of input terminals. Let $w_{j1}, w_{j2}, ..., w_{jp}$ denote the corresponding synaptic

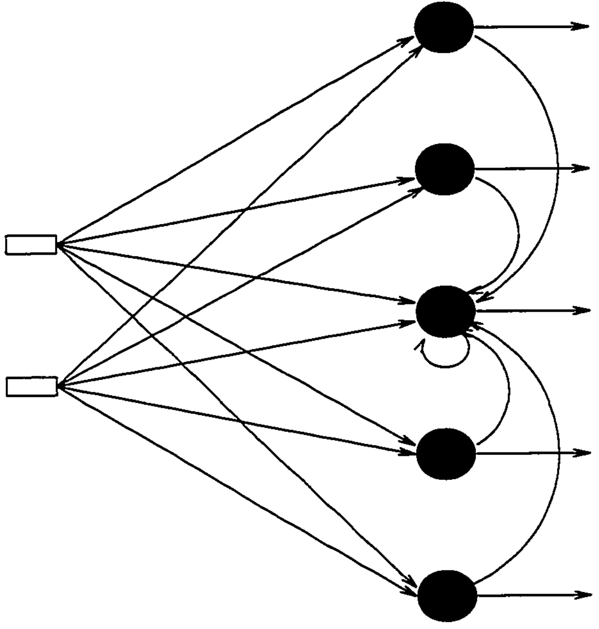


Figure 7.4: One-dimension lattice of neurons with feedforward connections and lateral feedback connections: the latter connections are shown only for the neuron at the center of array.

weights of neuron j. Let $c_{j,-K},\ldots,c_{j,-1},c_{j,0},c_{j1},\ldots,c_{jK}$ denote the lateral feedback weights connected to neuron j, where K is the "radius" of the lateral interaction. Let $y_1,y_2,\ldots,y_N$ denote the output signals of the network, where N is the number of neurons in the network. We may thus express the output signal of neuron j as follows:

$$y_j = \varphi\left(I_j + \sum_{k=-K}^{K} c_{jk} y_{j+k}\right), \quad j = 1,2,\ldots,N. \tag{7.1}$$

where $\varphi(.)$ is some nonlinear function that limits the value of y_j and ensures that $y_j \geq 0$. The term I_j serves the function of a stimulus, representing the total external control exerted on neuron j by the weighted effect of the input signals; that is,

$$I_j = \sum_{l=1}^{p} w_{jl} x_l.$$

The solution to the nonlinear Eq.7.1 is found iteratively, using a relaxation technique. Specifically, we reformulate it as a difference equation as follows:

$$y_j(n+1) = \varphi\left(I_j + \beta \sum_{k=-K}^{K} c_{jk} y_{j+k}(n)\right), \quad j = 1, 2, \ldots, N, \tag{7.2}$$

where n denotes discrete time. Thus $y_j(n+1)$ is the output of neuron j at time $n+1$, and $y_{j+k}(n)$ is the output of neuron $j+k$ at the previous time n. The parameter β in the argument on the right-hand side of Eq.7.2 controls the rate of convergence of the relaxation process.

The relaxation equation 7.2 represents a feedback system which includes both positive and negative feedback. The parameter β plays the role of a feedback factor of the system. The limiting action of the nonlinear activation function $\varphi(.)$ causes the spatial response $y_j(n)$ to stabilize in a certain fashion, dependent on the values assigned to β. If β is large enough, then in the final stage corresponding to $n \rightarrow \infty$, the values of y_j tend to concentrate inside a spatially bounded cluster, that is, an "activity bubble". The bubble is centered at a point where the initial response $y_j(0)$ due to the stimulus I_j is maximum. The width of the activity bubble depends on the ratio of the excitatory to inhibitory lateral interconnections. In particular, we may state the following:

- If the positive feedback is made stronger, the activity bubble becomes wider.
- If the negative feedback is enhanced, the activity bubble becomes sharper.

Of course, if the net feedback acting around the system is too negative, formation of the activity bubble is prevented.

7.2.3 Algorithm of SOFM

The self-organized feature-mapping (SOFM) model belongs to the class of vector coding algorithm [50], which was introduced by Kohonen. The SOFM network is comprised of two layers: a presynaptic neuron layer (input layer) and postsynaptic neuron layer (output layer). The input vector, representing the set of input signals, is denoted by

$$X = [x_1, x_2, ..., x_p]^T, \tag{7.3}$$

where p is the number of neurons in the presynaptic layer. The synaptic weight vector of neuron j in postsynaptic layer is denoted by

$$W_j = [w_{j1}, w_{j2}, ..., w_{jp}]^T, \quad j = 1, ..., N, \tag{7.4}$$

where N is the number of neurons in the postsynaptic layer. The input vector X is connected to each postsynaptic neuron by a feed forward synaptic weight vector. In the postsynaptic layer, there exist self-feedback and lateral feedback to control the postsynaptic neuron activity level. If the feedbacks are appropriately introduced, the network tends to concentrate its electrical activity into a local cluster, referred to as an activity bubble. The locations of the activity bubbles are determined by the nature of the input signals. To find the best match of the input vector X with the synaptic weight vector W_j , we simply compare the inner products $W_j^T X$ for $j = 1, ..., N$ and select the largest amongst them. This assumes that the same threshold is applied to all the neurons. Thus, by selecting the neuron with the largest inner product $W_j^T X$, we will have in effect identified the location of the activity bubble.

In the formulation of the adaptive algorithm, we find it convenient to normalize the weight vector W_j to constant Euclidean norm (length). In such a situation, the criterion for best match is equivalent to selecting the minimum Euclidean distance between vectors. Specifically, if we use the index $i(X)$ to identify the neuron that best matches the input vector X, we may then determine $i(X)$ by applying the condition:

$$i(X) = \arg\min_j \|X - W_j\|, \quad j = 1, ..., N, \tag{7.5}$$

where $\|.\|$ denotes the Euclidean norm of the argument vector. The particular neuron i that satisfies this condition is called the best-matching or winning neuron for the input vector X. By using Eq.7.5, a continuous input space is mapped onto a discrete set of neurons. Depending on the application of interest, the response of the network can either be the index of the neuron, or the synaptic weight vector.

For the network to be self-organizing, the synaptic vector W_j of neuron j is required to change in relation to the input vector X. In Hebb's postulate of learning, a synaptic weight is increased with a simultaneous occurrence of presynaptic and postsynaptic activities. For the type of unsupervised learning being considered here, however, the Hebbian hypothesis in its basic form is unsatisfactory for the following reason: changes in synaptic weights occur in one direction only, which finally drive all the weights into saturation. To overcome this problem, we may modify the Hebbian hypothesis simply by including a nonlinear forgetting term $-g(y_j)W_j$, where W_j is the synaptic weight vector of neuron j, and $g(y_j)$ is some positive scalar function of its response y_j. The only requirement imposed on the function $g(y_j)$ is that the constant term in the Taylor series expression of $g(y_j)$ be zero, so that we may write

$$g(y_j) = 0 \quad \text{for } y_j = 0 \text{ and for all } j \ .$$

The significance of this requirement will become apparent momentarily. Given such a function, we may then express the differential equation that defines the computational map produced by the SOFM algorithm as

$$\frac{dW_j}{dt} = \eta y_j X - g(y_j)W_j, \quad j = 1, 2, ..., N, \tag{7.6}$$

where t denotes the continuous time and η is the learning-rate parameter of the algorithm.

Let $\Lambda_{i(X)}$ denote the topological neighbourhood of the winning neuron, $i(X)$. When the feedback is strong enough in the postsynaptic layer, the neurons inside the topological neighbourhood of winning neuron are saturated due to the clustering effect (i.e., the formation of an activity bubble). The neuron response is "one" if the neuron is inside the neighbourhood $\Lambda_{i(X)}$, otherwise it is "zero". Furthermore, let $g(y_j) = \alpha$ when jth neuron is active, where α is a positive constant. Without loss of generality, we may put $\alpha = \eta$, in which case Eq.7.6 simplifies as

$$\frac{dW_j}{dt} = \begin{cases} \eta(X - W_j), & j \in \Lambda_{i(X)} \\ 0 & \text{otherwise.} \end{cases} \tag{7.7}$$

Finally, using discrete-time formation, Eq.7.7 is cast in a form whereby, given the synaptic weight vector $W_j(n)$ of neuron j at discrete time n, we may compute the updated value $W_j(n+1)$ at time $n+1$ as follows [50]

$$W_j(n+1) = \begin{cases} W_j(n) + \eta(n)[X(n) - W_j(n)], & j \in \Lambda_{i(X)}(n) \\ W_j(n) & \text{otherwise.} \end{cases} \tag{7.8}$$

Here $\Lambda_{i(X)}(n)$ is the neighbourhood around the winning neuron i at time n, and $\eta(n)$ is the corresponding value of the learning rate parameter. The neighbourhood $\Lambda_{i(X)}(n)$ is chosen to be a function of the discrete time n; hence we may also refer to $\Lambda_{i(X)}(n)$ as a neighbourhood index function, which reflects the bubble size. Numerous simulations have shown that the best results in self-organization are obtained if the neighbourhood index function $\Lambda_{i(X)}(n)$ is selected fairly wide in the beginning and then permitted to shrink with time n [50]. This behavior is equivalent to initially using a strong positive lateral feedback, and then enhancing the negative lateral feedback. The important point to note here is that the use of a neighbourhood index function $\Lambda_{i(X)}(n)$ around the winning neuron $i(X)$ provides a clever computational shortcut for emulating the formation of a localized response by lateral feedback. This idea is the motivation for using the adaptive bubble size.

There are three basic steps involved in the application of the algorithm after initialization, namely, sampling, similarity matching, and updating. These three steps are repeated until the map formation is completed. The algorithm is summarized as follows [50]:

1. **Initialization**: choose random values for the initial weight vectors $W_j(0)$, $j = 1, 2, ..., N$. It may be desirable to keep the magnitude of the weights small.

2. **Sampling:** draw a sample X from the input distribution with a certain probability; the vector X is the input of network.

3. **Similarity matching:** find the best matching neuron $i(X)$ at time n using the following criterion: $i(X) = \arg\min_j \|X(n) - W_j\|, j = 1, \ldots, N$.

4. **Updating:** adjust the synaptic weight vectors of all neurons using the update formula:

$$W_j(n+1) = \begin{cases} W_j(n) + \eta(n)[X(n) - W_j(n)], & j \in \Lambda_{i(X)}(n) \\ W_j(n) & \text{otherwise}, \end{cases} \tag{7.9}$$

   where $\eta(n)$ is the learning-rate parameter, and $\Lambda_{i(X)}(n)$ is the neighbourhood index function centered around the winning neuron $i(X)$; both η and $\Lambda_{i(X)}$ are varied dynamically during learning for best results.

5. **Continuation:** continue with *step* 2 until no noticeable changes in the feature map are observed.

The learning process involved in the computation of a feature map is stochastic in nature, which means that the accuracy of the map depends on the number of iterations of the SOFM algorithm. Moreover, the success of map formation is critically dependent on how the main parameters of the algorithm, namely, the learning-rate parameter η and the neighborhood index function Λ_i, are selected. There is no theoretical basis for the selection of these parameters. They are usually determined by a process of trial and error. Nevertheless, the following observations provide a useful guide.

1. The learning-rate parameter $\eta(n)$ used to update the synaptic weight vector should be time-varying. In particular, during first 500 iterations or so, $\eta(n)$ should begin with a value close to unity; thereafter, $\eta(n)$ should decrease gradually, but staying above 0.1. The exact form of variation of $\eta(n)$ with n is not critical; it can be linear, exponential, or inversely proportional to n. It is during this initial phase of the algorithm that the topological ordering of the weight vector $W_j(n)$ takes place. This phase of the learning process is called the order phase. The remaining iterations of the algorithm are needed principally for the fine tuning of the computation map; this second phase of the learning process is called the convergence (stable) phase. For good statistical accuracy, $\eta(n)$ should be maintained during the convergence phase at a small value (on the order of 0.01 or less) for a fairly long period of time, which is typically thousands of iterations.

2. For topological ordering of the weight vector W_j to take place, carefully consideration has to be given to the neighborhood index function Λ_i. It can take any shape. In any case, the neighborhood function Λ_i usually begins such that it includes all neurons in the network and then gradually shrinks with time. To be specific, during the initial phase of 500 iterations or so, when topological ordering of the synaptic weight vectors takes place, the radius of Λ_i is permitted to shrink linearly with time n to a small value of only a couple of neighboring neurons. During the convergence phase of the algorithm, Λ_i should contain only the nearest neighbors of winning neuron i.

7.3 Measurement Data Fusion

In Eq.7.9. the bubble corresponding to the neighbourhood index function around the winning neuron i (this bubble is denoted as the neighbourhood function) is assumed to have a constant amplitude. The implication of this model is that all the neurons located inside this topological neighbourhood fire at the same rate, and interaction among those neurons is independent of their lateral distance from the winning neuron i. However, from a neurobiological viewpoint there is evidence for lateral interaction among neurons in the sense that a neuron that is firing tends to excite the neurons in its immediate neighbourhood more than those farther away from it. To account for the lateral effect of a winning neuron i on the activity of its neighbouring neurons, we may make the topological neighbourhood function around winning neuron, i, decay with lateral distance.

Let d_{ji} denote the lateral distance of neuron j from the winning neuron i, which is a Euclidean measure in output space. Let $\bigcap_{ji}$ denote the neighbourhood function centered on the winning neuron i. From the discussion on neurobiological evidence, the $\bigcap_{ji}$ is chosen:

$$\bigcap_{ji} = \exp(-\frac{d_{ji}^2}{2\sigma^2}), \tag{7.10}$$

where the parameter σ is the "effective width" of the topological neighbourhood. This is a Gaussian neighbourhood function. Therefore we may now rewrite the recursive relation (Eq.7.9) for updating the synaptic weight vector W_j of neuron j at lateral distance d_{ji} from the winning neuron $i(X)$ as follows:

$$W_j(n+1) = W_j(n) + \eta(n)\bigcap_{ji(X)}(n)[X(n) - W_j(n)], \tag{7.11}$$

where the learning-rate parameter $\eta(n)$ and the neighbourhood function $\bigcap_{ji(X)}(n)$ are both shown as being dependent on time n. For the SOFM algorithm to converge, it is important that the width $\sigma(n)$ of the topological neighbourhood, as well as the learning rate parameter $\eta(n)$, be permitted to decrease slowly during the learning process. A popular choice for the dependence of $\sigma(n)$ and $\eta(n)$ on time n is the exponential decay described as, for $\sigma(n)$ and $\eta(n)$ respectively,

$$\sigma(n) = \sigma_o \exp(-\frac{n}{\tau_1}), \tag{7.12}$$

and

$$\eta(n) = \eta_o \exp(-\frac{n}{\tau_2}). \tag{7.13}$$

The constant σ_o and η_o are the respective values of $\sigma(n)$ and $\eta(n)$ at the initialization of SOFM (i.e., at $n = 0$), and τ_1 and τ_2 are their respective time constants, which are determined according to data category. Eqs.7.12, and 7.13 are applied to the neighbourhood function and the learning rate parameter, respectively, only during the ordering phase. In the stable learning phase, the small values are assigned to these parameters for refining the mapping results.

It is assumed that there are L targets and M radar sets. We can get P ($P = L \times M$) position observations each scan, with each observation having two components. We suppose that coordination transformation and synchronization have been done. Here we have L predicted points. Based on the above assumptions, the feature-mapping net can be established with two neurons in the presynaptic layer and $P + L$ neurons in the postsynaptic layer, where L neurons correspond to the predicted points, which are forced to their firing states. P neurons correspond to the observations. The topologically lateral distance of the jth neuron corresponding to the jth observation from the kth firing neuron corresponding to the kth target is a Euclidean measure of the jth observation with respect to the kth predicted point. The outputs of the neural network are the synaptic weights which are connected to winning neurons. The whole learning process is an adaptive one. There are two phases in the adaptive process: an ordering phase and that which is referred to as the stable phase. In the ordering phase, the effective width of the topological neighbourhood $\sigma(n)$, and learning rate $\eta(n)$ are changed in the iterative process according to Eqs.7.12, 7.13. In the stable phase, the $\sigma(n)$ and $\eta(n)$ are assigned in appropriate constant value. The choice of $\eta(n)$ will affect the speed of convergence and the choice of $\sigma(n)$ is dominated by the performance of a position predictor.

In the computer simulations that follow, we normalize the radar plan position indicator (PPI): the X-axis varies from -5 to 5 and the Y-axis covers the same range of values. The departures of the observation point from the tracks are dominated by Gaussian random processes. The extent to which the observations are scattered is controlled by the precisions of the radars. In Fig.7.5 is shown a one-target fusion result using the feature-mapping neural network presented here. The data are obtained using four radars, all with the same precision: $\sigma = 1$. In Fig.7.6 is demonstrated a one-target feature-mapping fusion case using ten sensors, each with the same precision,

$\sigma = 1$. It can be seen that the more the number of sensors we use, the more accurate the fusion result. In Fig.7.7 are given the feature-mapping fusion results for two targets with ten sensors, where each sensor's precision σ is equal to 0.8. In Fig.7.8 are presented the fusion results for two targets with ten sensors, each having precision $\sigma = 0.8$. Here we can see that the two observation clusters overlap each other; it is very difficult to separate them. However feature-mapping fusion does not need to know the cluster boundaries nor the measurement error distributions, it learns from the observations. In practice, this is a significant advantage over classical methods.

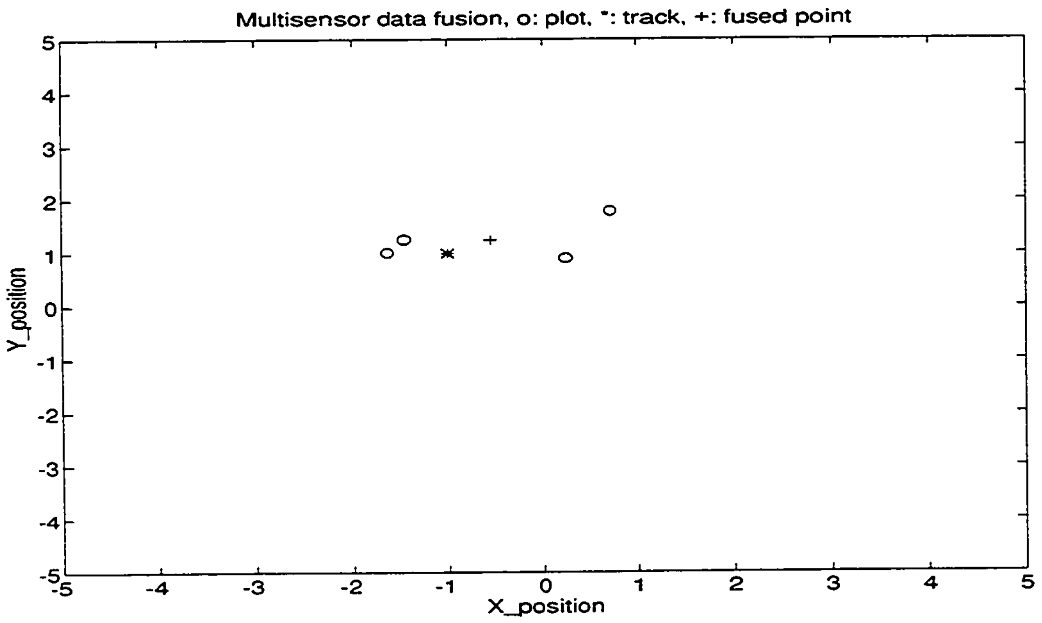


Figure 7.5: Feature-mapping data fusion result for one target and four sensors, sensor precision $\sigma = 1$.

In the stable learning phase, the weight vector of the winning neuron would not change any more after a certain number of iterations. At this point, all the

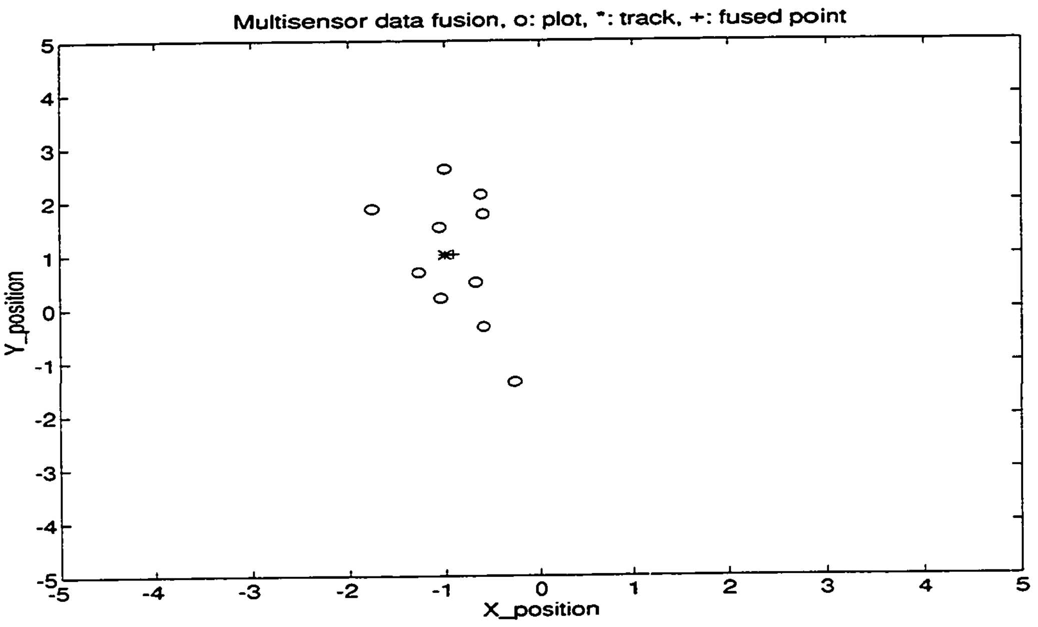


Figure 7.6: Feature-mapping data fusion result for one target and ten sensors, sensor precision $\sigma = 1$.

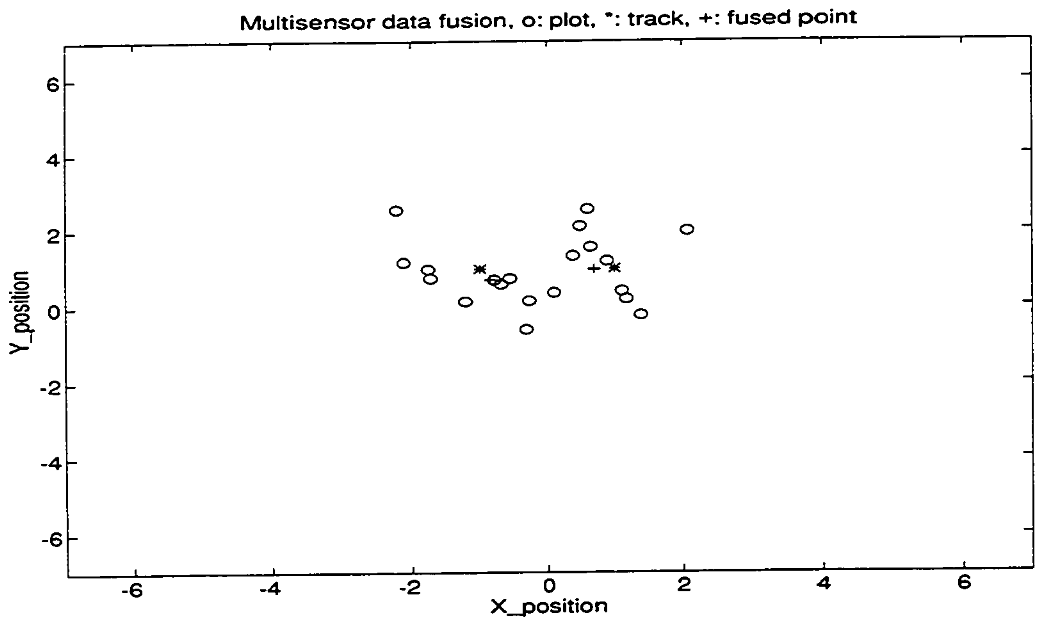


Figure 7.7: Feature-mapping data fusion results for two targets and ten sensors, sensor precision $\sigma = 0.8$.

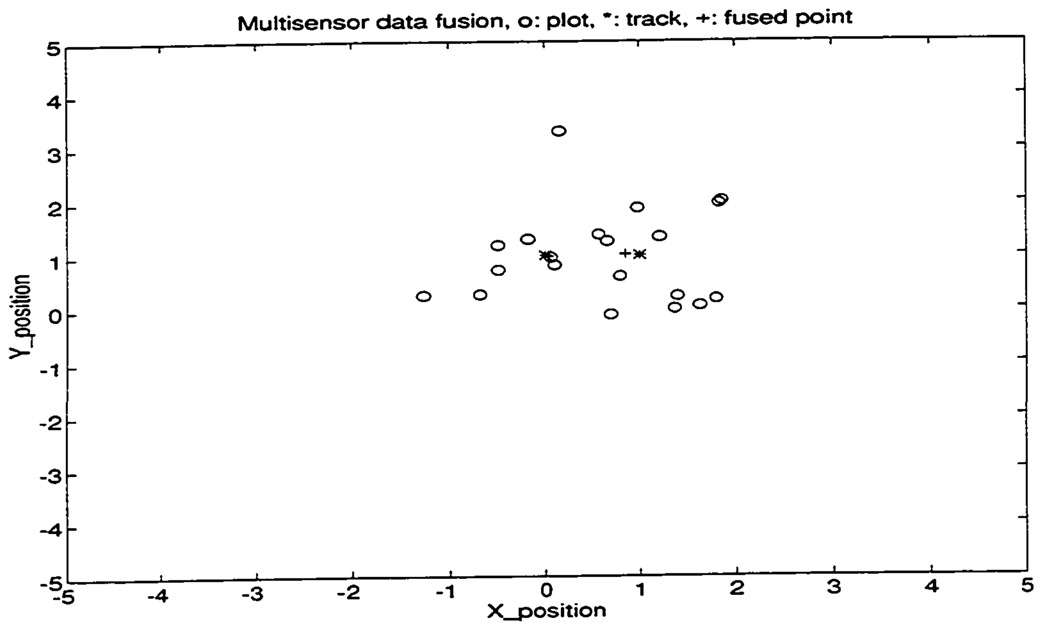


Figure 7.8: Feature-mapping data fusion results for two targets and ten sensors, sensor precision $\sigma = 0.8$.

knowledge supplied by the plots in a given scan of measurement data is learned by the network. In the above methodological development, we used the two-dimensional measurement (position (X,Y)). It is easy to extend it to multiple dimensional case. If the target velocity is available, it can be incorporated to enhance the fusion ability. The feature-mapping net can be established with four neurons in the presynaptic layer, with one for perceiving the speed, one for acceleration, and two for position. It could track the manoeuvring target. In practice, the Kalman filter is used as a position predictor, its output is related to the history of a track, and this output is used to determine the lateral feedback between the neuron standing for predicting point and that denoting observation in the postsynaptic layer. Therefore the track history has been sensed by the network through a position predictor in the learning process.

Based on the learning rule, Eq.7.11 can be applied to any data distribution model, therein allowing for the determination of the centroid. From the method development above, it can be seen that the network does not involve recurrent process, the learning is based on the comprehensively modified Hebbian postulate. Therefore, the learning speed is fast. As mentioned above, SOFM learning does not need to know the prior knowledge of sensors' accuracies. Nevertheless, if such data is available, it can be deployed to determine the time constants τ_1 and τ_2, the initial value of the effective width of the topological neighbourhood, σ_0, and the initial learning rate η_0, the learning will be optimized. For example, when $\sigma = 1$, the learning parameters are set as $\tau_1 = 200$, $\sigma_0 = 200$, $\tau_2 = 100$, $\eta_0 = 1$, the number of iteration in the ordering phase is 500. However, when $\sigma = 0.6$, we reduce the σ_0 in half, and decrease τ_1 in half, the other parameters keep unchanged, the number of iteration in the ordering phase is 200.

7.4 Comparison of Fusion Accuracy

Maximum likelihood (ML) is a statistical method whose estimation accuracy can rearch the Cramer-Rao lower bound in the case of Gaussian distributed process. On the other hand, the SOFM network is based on unsupervised learning, which is inspired by a study of neurobiology. In the following, we first investigate ML, then compare data fusion performances of the SOFM network and conventional methods.

7.4.1 Maximum Likelihood Approach

Multisensor data fusion is used to estimate the target's dynamic centroid. It performs a function that is exactly like that of a parameter estimator. A class of estimators that are often used are the maximum likelihood estimators. The method of maximum likelihood is based on a relatively simple idea: Different populations generate different data samples and any given data sample is more likely to have come from some population than from others.

Let $p_X(X/\Phi)$ denote the conditional joint probabilistic density function (PDF) of the random vector X having $x_1, ..., x_M$ as its elements, and Φ be a parameter vector with $\phi_1, ..., \phi_K$ as its elements. The method of ML is based on the principle that we should estimate the parameter vector by its most plausible values, given the observed sample vector X. In other words, the ML estimates of $\phi_1, ..., \phi_K$ are those values of the parameter vector for which $p_X(X/\Phi)$ is at maximum.

The likelihood function, denoted by $l(\Phi)$, is given by the conditional joint PDF $p_X(X/\Phi)$, viewed as a function of the parameter vector Φ. Thus

$$l(\Phi) = p_X(X/\Phi).$$

Although the conditional joint PDF and the likelihood function have exactly the same formula, nevertheless, it is essential to appreciate their distinction. In the case of conditional joint PDF, the parameter Φ is fixed and the observation vector is variable. On the other hand, in the case of likelihood function, the parameter vector Φ is variable and the observation vector X is fixed.

Very often, it turns out that it is more convenient to work with the logarithm of the likelihood function rather than with the likelihood function itself. Thus, using $L(\Phi)$ to denote the log-likelihood function, we write,

$$L(\Phi) = \ln[l(\Phi)] = \ln[p_X(X/\Phi)].$$

The logarithm of $l(\Phi)$ is a monotonic function of $l(\Phi)$. This means whenever $l(\Phi)$ increases, its logarithm $L(\Phi)$ also increases. Thus, maximizing $l(\Phi)$ is equivalent to maximizing $L(\Phi)$. The ML estimate $\hat{\Phi}$, obtained through maximizing $L(\Phi)$, satisfies the Cramer-Rao lower bound for joint Gaussian process, which is one of the best estimates.

In the application of ML approach to multiple sensor data fusion, it is supposed that there are N independent radar measurements ξ_i's from N radar sites for each target with its centroid being a_j, and that ξ_i belongs to Gaussian distribution, the log-likelihood function of a_j can be expressed as

$$L(a_j) = \ln \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_i} + \sum_{i=1}^{N} \frac{-(\xi_i - a_j)^2}{2\sigma_i^2}, \tag{7.14}$$

where σ_i is the precision of *ith* measurement radar. The solution of maximizing $L(a_j)$ can be easily obtained as follows:

$$\mathrm{a_j} = \frac{\sum_{i=1}^{N} \frac{\xi_i}{\sigma_i^2}}{\sum_{i=1}^{N} \frac{1}{\sigma_i}}. \tag{7.15}$$

Eq.7.15 is used for multiple sensor data fusion, and its performance is compared with that of the SOFM data fusion in the following subsection.

7.4.2 Performance Evaluation

Using computer simulations, we compare the fusion performances obtained using feature-mapping network, Maximum Likelihood (ML) and *ad hoc* methods. In the comparisons, we consider seven sensors with measurement accuracies (standard deviation $\sigma's$) 2, 0.5, 1, 0.3, 1.8, 1.4, 1, respectively. In each case, both the number of sensors and the measurement accuracies are specified. The simulations for each case are run 1000 times to calculate the variance of the fused point for each method. For the ML method, we assume that each radar's accuracy is known, otherwise the computational overload is huge. The results of calculation are plotted against the number of sensors. In Fig.7.9 is presented the fusion performance for the three methods. It can be seen that the performance of the feature-mapping neural network (Koh fusion in the figure) is as good as that for ML, and much better than that for the *ad hoc* method. In Fig.7.10 are given the fusion variances of the feature-mapping and ML methods, calculated for the same parameters used in Fig.7.9, except that here each sensor's precision is the same. Based on these results, it is obvious that the greater the number of sensors is, the better the fusion results. The feature-mapping fusion approach enjoys many advantages over ML. As mentioned before, it learns from the observations, and therefore it does not need to have the prior knowledge regarding the sensors nor the prior information on the measurement error distributions. As well, the fusion speed is fast.

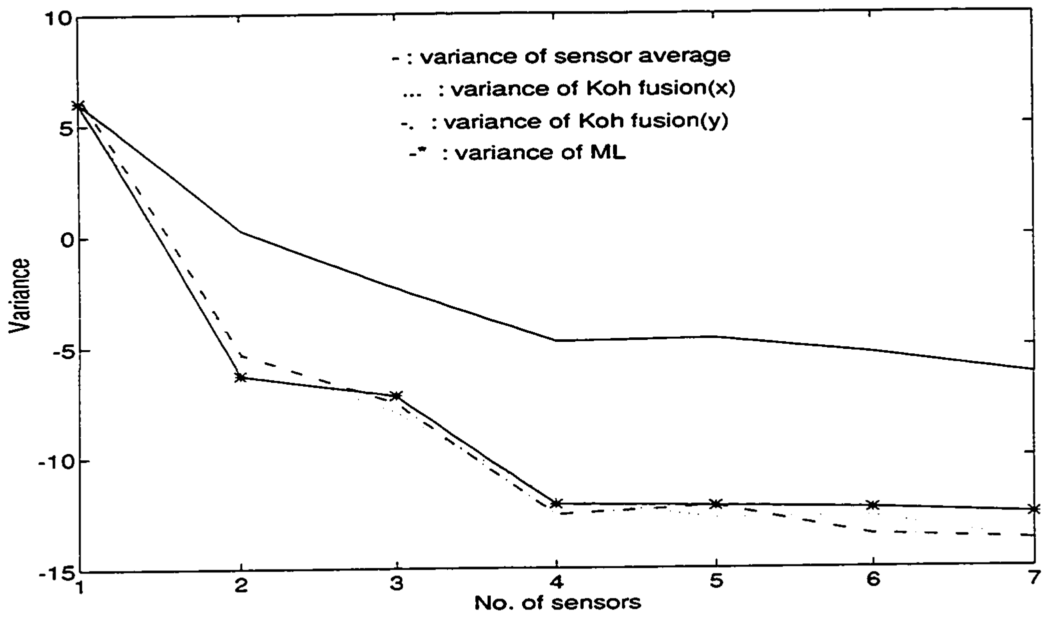


Figure 7.9: Variance comparison for different fusion methods.

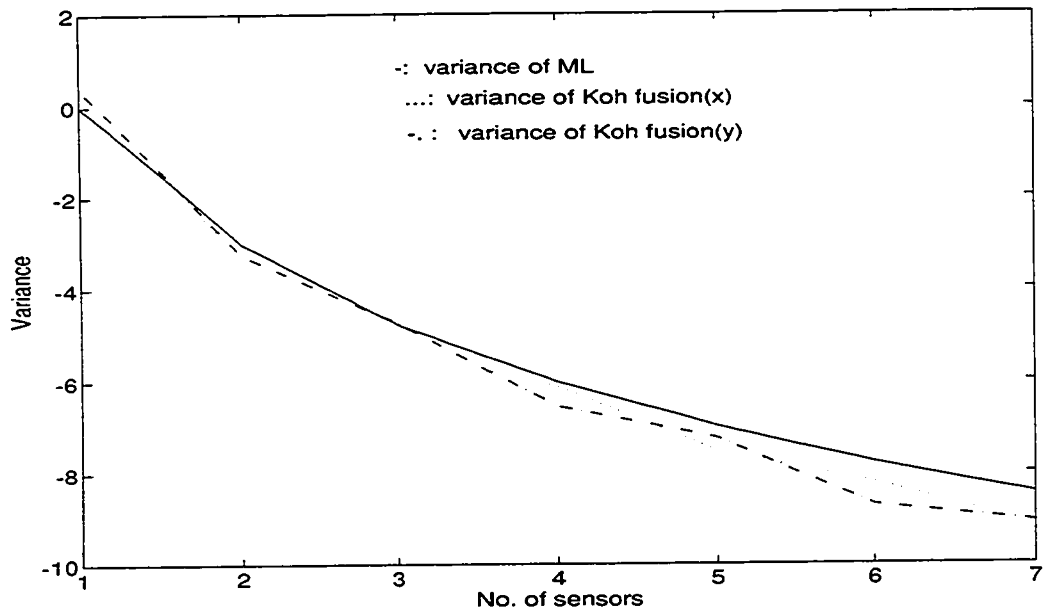


Figure 7.10: Variance comparison for different fusion methods, $\sigma = 1$.

7.5 Summary

We conclude that it is feasible to use the feature-mapping data fusion in practice. Since this is based on a neural network technique, it can learn from the observations and get the same precise fusion centroid as that obtained by applying ML in the case of Gaussian distributed measurement error. As we know, in the case of ML, the measurement error distribution function and sensors accuracies need to be known a priori. If the sensors' accuracies are not available, the calculation overload is huge. If the error distribution function is unknown, ML can do nothing for the fusion problem. Fortunately, the feature-mapping fusion approach overcomes the drawbacks of ML. On the other hand, the method presented here can be easily built into a VLSI network which possesses parallel computation potency. In other methods, as well, the cluster boundary should be found in order that the data, which are from the same target, can be fused. In a dense target environment, the feature-mapping fusion approach avoids the difficulty of finding the cluster boundary by introducing adaptive lateral feedback in the postsynaptic layer. This makes the fusion implementation very feasible and easy.

Chapter 8

Conclusions

8.1 Contributions

In this thesis. we have discussed neural networks and multisensor data fusion. and developed the techniques for multisensor target classification. Multilayer perceptrons neural networks trained by Backpropagation have been designed. The whole system. like hierarchical expert network structures, is easily trained using the adaptive learning rate and momentum methods. This hierarchical network configuration has a capacity to classify the targets.

A new technique of measurement data association (MFTDA) in a multitarget radar environment is developed. The technique is based on the mean-field-theory machine and has the advantages of both the Hopfield network and the Boltzmann machine. The new energy function built in the thesis can be considered a cost function. The cost function takes a minimal value when the plot-track association assumes its optimal status. In the technical development, three new energy functions have been created. Theoretically, the critical annealing temperature is found to determine the annealing temperature range. Neural data association capacities have been evaluated in the cases with and without clutter for different measurement radar precisions.

As a part of the development of the neural data association, the convergent theorem established herein provides an affirmative assertion that the mean-field-theory machine globally asymptotically converges to a stable equilibrium provided the network satisfies the conditions described in the theorem. It also gives us a rule for devising physical energy function.

As a benchmark, the nearest neighbour association is compared with the neural data association. The neural data association has higher capacity than does the nearest neighbour association, it demonstrates remarkable improvement in high target density circumstance.

The new energy functions have been extended to multiple dimensional data association. A comprehensive analysis by computer simulations has demonstrated that the new technique (MFTDA) developed in the thesis possesses high association capacity in the presence of false alarms: it can cope with track-crossing in the dense target environment. In the case $\sigma_p = 0.3$, the neural data association capacity with multiple dimensional measurements is 30% higher than when only using position measurements for the environment described in Chapter 5.

The feature-mapping data fusion (FMDF) presented in the thesis has much more advantages over the conventional fusion approach. Since it is based on a neural network technique, it can learn from the observations and get the same precise fusion centroid as that obtained by applying ML in the case of Gaussian distributed measurement error. As we know, in the case of ML, the measurement error distribution function and sensors accuracies need to be known a priori. If the sensors accuracies are not available, the calculation overload is huge. If the error distribution function is unknown, ML can do nothing for a fusion problem. Fortunately, the feature-mapping fusion approach overcomes the drawbacks of ML. On the other hand, the method presented here can easily be built into a VLSI network which possesses parallel computation potency. In other methods, as well, the cluster boundary should be found in order that the data, which are from the same target, can be fused. In a dense target environment, the feature-mapping fusion approach avoids the difficulty inherent to finding the cluster boundary by introducing adaptive lateral feedback in the postsynaptic layer. This makes the fusion implementation very feasible and easy.

Based on the above results, we have generated two technical research reports for Defense Research Establishment Valcartier, Canada, and published four papers in well-known international conferences. Two journal papers have appeared in IEE Proc.- Radar, Sonar and Navigation, and another journal paper is in the course of review for publishing on IEEE AES.

8.2 Future Works

Future research will be carried out in the following areas:

1. Both techniques PDA and JPDA probabilistically "smooth" or "filter" the data within the gate of interest. The predicted track positions are associated with the smoothed points instead of real radar contacts. In MDA, the predicted track positions are associated with real radar contacts. Therefore the comparison between MDA and PDA or JPDA is not feasible at the association stage. In future study, we should compare them by track quality.

2. Based on the method development and its performance evaluation for FMDF, a VLSI neural network should be built for centralized data fusion system.

3. Hierarchical expert networks for recognition and classification should be investigated; it is possible that this sort of network has a potential in data fusion applications.

4. Hybrid networks for extended reasoning should be studied for situation assessment.

Bibliography

[1] Edward Waltz, and James Llinas, **Multisensor Data Fusion,** Artech House, Boston, 1990.

[2] David L. Hall, **Mathematical Techniques in Multisensor Data Fusion,** Artech House, Boston. 1992.

[3] Y. Bar-Shalom, and T.E. Fortmann, **Tacking and Data Association,** New York: Academic Press, 1988.

[4] Y. Bar-Shalom. and E. Tse, "Tracking in a cluttered environment with probabilistic data association," Automatica, 11, pp.451-460, Sept., 1975.

[5] T.E. Fortmann, Y. Bar-Shalom, and M. Scheffe, "Sonar tracking of multiple targets using joint probabilistic data association," IEEE Journal of Oceanic Engineering, OE-8, pp.173-184, July, 1983.

[6] D.B. Reid, "An algorithm for tracking multiple targets," IEEE Transactions on Automatic Control, AC-24, pp.843-854, Dec., 1979.

[7] Y. Bar-Shalom, **Multitarget-Multisensors Tracking: Advanced Applications,** Norwood, MA: Artech House, 1990.

[8] A.K. Mahalanabis. B. Zhou, and N.K. Bose, "Improved multitarget tracking in clutter by PDA smoothing," IEEE Transactions on Aerospace and Electronic System, 26. pp.113-121, Jan., 1990.

[9] R.J. Fitzgerald. "Development of practical PDA logic for multitarget tracking by microprocessor," In Proceedings of the American Controls Conference. Seattle. WA. pp.889-898, June. 1986.

[10] D. Sengupta, and R.A. Iltis. "Neural solution to the multiple target tracking data association problem," IEEE Transactions on Aerospace and Electronic System. 25. pp.96-108. Jan., 1989.

[11] B. Zhou. and N.K. Bose. " A comprehensive analysis of 'neural solution to the multiple target tracking data association problem,' " IEEE Transactions on Aerospace and Electronic Systems. 29, pp.260-263. Jan.. 1993.

[12] B. Zhou. and N.K. Bose. " A unified approach to data association in multitarget tracking," Automatica, Vol.30, no.9, pp.1469-1472. 1994.

[13] B. Zhou. and N.K. Bose. " Multitarget tracking in clutter: fast algorithms for data association," IEEE Transactions on Aerospace and Electronic Systems, Vol.29. No.2, pp.352-363. April, 1993.

[14] B. Zhou. and N.K. Bose, " An efficient algorithm for data association in multitarget tracking," IEEE Transactions on Aerospace and Electronic Systems, Vol.31, no.1. pp.458-468. Jan., 1995.

[15] J.L. Fisher, and D.P. Casasent, "Fast JPDA multitarget tracking algorithm," Applied Optics, 28, pp.371-376, Jan., 1989.

[16] V. Nagarajan, M.R. Chidambara, and R.N. Sharma, "Combinatorial problems in multitarget tracking- A comprehensive solution," IEE Proceedings Part F, Communications, Radar and Signal Processing, 134, pp.113-118, Feb., 1987.

[17] William F. Smith, Leon K. Ekchian, and David D. Johnson, "Neural network implementation of plot/track association," SPIE, April 1990.

[18] A.H. Gee, "Problem solving with optimization networks," Ph.D. dissertation, University of Cambridge, UK. 1993.

[19] Daniel Avitzour, "A maximum likelihood approach to data association," IEEE Transactions on Aerospace and Electronic Systems, Vol.28, No.2, April 1992.

[20] Jon A. Benediktsson, Philip H. Swain, and Okan K. Ersoy, " Neural network approaches versus statistical methods in classification of multisource remote sensing data," IEEE Transactions on Geoscience and Remote Sensing, Vol.28, No.4, July 1990.

[21] J.A. Anderson and E. Rosenfeld (Eds.), **Neurocomputing Foundation of Research,** Cambridge: The M.I.T. Press, 1988.

[22] W.S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, Vol.5, pp.115-133, 1943.

[23] K. Fukushima, "Cognitron: a self organizing multilayered neural network," Biolog. Cybernetics, Vol.20, pp.121-136, 1975.

[24] L. Herault, J.J. Niez, "Neural networks and combinatorial optimization, a study of NP-complete graph problems," in Neural networks: Advances and Applications, Editor E. Gelenbe, Elsevier, Amsterdam, 1991, pp.165-213.

[25] E.L. Lawler, J.K. Lenstra, A.H. Rinnooy Kan and D.B. Shmoys (Editors), **The Travelling Salesman Problems; A Guided Tour of Combinatorial Optimization,** Wiley, New York, 1985.

[26] P.M. Pardalos and S. Jha, "Graph separation techniques for quadratic zero-one programming," Computers Math. Applic., Vol.21, 1991, pp.107-113.

[27] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," Bulletin of the American Mathematical Society, 64, pp.275-278, 1958.

[28] E.L. Lawler and D.E. Wood, "Branch-and-bound methods: a survey," Operations Research, 14, pp.699-719, 1966.

[29] S.E. Dreyfus and A.M. Law, **The Art and Theory of Dynamic Programming,** Academic Press, 1977.

[30] E.M. Reingold, J. Nevergelt, and N. Deo, **Combinatorial Algorithms–Theory and Practice,** Prentice Hall, New Jersey, 1977.

[31] M. R. Garey and D.S. Johnson, **Computers and Intractability–A Guide to the Theory of NP-Completeness,** W.H. Freeman and Company, San Fransisco, 1979.

[32] M.L. Balinksi, **Approaches to Integer Programming,** Mathematical Programming Studies. North Holland Publishing Company, Amsterdam, 1974.

[33] S. Lin and B.W. Kernighan, "An efficient heuristic algorithm for the traveling salesman problem," Operations Research, 21(2), pp.498-516, 1973.

[34] D.W. Tank and J.J. Hopfield, "Simple 'neural' optimization network: an A/D converter, signal decision circuit and a linear programming circuit," IEEE Trans. Circuits and Systems, Vol.CAS-33, pp.533-541, 1986.

[35] J.J. Hopfield and D.W. Tank, "Neural computation of decisions in optimization problems," Biolog. Cybernetics, Vol.52, pp.141-152, 1985.

[36] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proc. Natl. Acad. Sci., Vol.79, pp.2554-2558, 1982.

[37] J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," Proc. Natl. Acad. Sci., Vol.81, pp.3088-3092, 1984.

[38] J.H. Li, A.N. Michel and W. Parod, "Analysis and synthesis of a class of neural networks: linear systems operating on a closed hypercube," IEEE Trans. Circuits and Systems, Vol.CAS-36, pp.1405-1422, 1989.

[39] J. Bruck, "On the convergence properties of the Hopfield model," Proc. IEEE (Special issue on neural networks analysis, techniques and applications, Eds. C. Lau and B. Widrow), pp.1579-1585, 1990.

[40] E. Aart and J. Korst, **Simulated Annealing and Boltzmann Machines,** New York: Wiley, 1989.

[41] D.E. Van den Bout and T.K. Miller, III, "A digital architecture employing stochasticism for the simulation of Hopfield neural nets," IEEE Trans. Circuits and Systems, Vol.36, No.5, pp.732-738, 1989.

[42] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing," Science, Vol.220, pp.671-680, 1983.

[43] S. Kirkpatrick, "Optimization by simulated annealing: quantitative studies," Journ. Statist. Physics, Vol. 34, pp.974, 1984.

[44] J.P. Cerny, "Thermodynamical approach to the traveling salesman problem," Journal of Optimization Theory and Applications, Vol.45, pp.41-51, 1985.

[45] N. Metropolis, A. Rosenbluth, M. Rosenbluth, M. Teller, and E. Teller, "Equation of state calculations by fast computing machine," J. Chemical Physics, Vol.21, No.6, pp.1087-1092, 1953.

[46] T.J. Sejnowski, "Neural network learning algorithms," Neural computers, Springer-Verlag, pp.291-300, 1988.

[47] C. Peterson and J.R. Anderson, "A mean field learning algorithm for neural networks," Complex Systems, Vol.1, pp.995-1019, 1987.

[48] C. Peterson and B. Soderberg, "A new method for mapping optimization problems onto neural networks," Int. Journal of Neural Systems, Vol.1, No.1, pp.3-22, 1989.

[49] C. Peterson, and E. Hartman, "Exploration of the Mean Field Theory Learning Algorithm," Neural Networks, Vol.2, pp.475-494, 1989.

[50] Simon Haykin, **Neural Networks: a Comprehensive Foundation,** Macmillan College Publishing Company, INC., New York, 1994.

[51] D.E. Van den Bout, and T.K. Miller III, "Improving the performance of the Hopfield-Tank neural network through normalization and annealing," Biol. Cybern. 62, pp.129-139, 1989.

[52] C.A. Noonan, M.E. Everett, and R.C. Freeman, "Sensor data fusion for air to air situation awareness beyond visual range," AGARD conference proceedings, 539, 1993.

[53] S.S. Blackman, **Multiple Target Tracking with Radar Applications,** Artech House, Dedham, Massachusetts, 1986.

[54] F. Wang, T. Lo, J. Litva, and E. Bosse, "A new energy function of mean field Hopfield network for measurement data association," ICSPAT'94, Dallas, USA, 1994.

[55] F. Wang, and John Litva, "Neural data association" ICNNSP'95, Nanjing, China, Dec., 1995.

[56] John Litva, F. Wang, and T. Lo, "Neural networks for multisensor data fusion," CRL Report NO. 298, Applied to DSS CONTRACT No.W7701-3-1426/01-XSK, Canada. 1995.

[57] John Litva, F. Wang, and T. Lo, "Neural networks for multisensor data fusion - A feature mapping approach," CRL Report, Applied to DSS CONTRACT No.W7701-3-1426/01-XSK, Canada. 1996.

[58] F. Wang, John Litva, T. Lo, and E. Bosse, "Performance of neural data associator," IEE Proc.- Radar, Sonar and Navig., Vol.143, No.2, pp.71-78, April 1996.

[59] F. Wang, and John Litva, "Feature Mapping Data Fusion," IEE Proc.- Radar, Sonar and Navig., Vol.143, No.2, pp.65-70, April 1996.

[60] F. Wang, John Litva, "Multidimensional neural data association," submitted to IEEE AES, 1996.

[61] T. Lo, F. Wang, H. Leung, and J. Litva, "Neural Networks for Multisensor Multitarget Tracking," 27th International Symposium on Automotive technology and Automation, Aachen, Germany, 1994.

[62] F. Wang, T. Lo, J. Litva, and E. Bosse, " Multisensor Automatic Target Classification with Neural Networks," Seventh SP Workshop on Statistical Signal and Array Processing, Quebec, Canada, 1994.

[63] R.A. Singer, and A. J. Kanyuck, "Computer control of multiple site correlation," Automatica, 7, pp.455-463, 1971.

[64] W.R. Ditzler, "A demonstration of multi-sensor tracking," Proceedings of the 1987 Tri-service Data Fusion Symposium, pp.303-311, 1987.

[65] Y. Bar-Shalom, and L. Campo, "The effect of the common process noise on the two-sensor fused-track covariance," IEEE Transactions on Aerospace and Electronic Systems, AES-22, pp.803-805, 1986.

[66] J.A. Roecker, and C.D. McGillem, "Comparison of two-sensor tracking methods based on state vector fusion and measurement fusion," IEEE Transactions on Aerospace and Electronic systems, AES-24, pp.447-449, 1988.

[67] T. Kohonen, "The self-organizing map," Proceedings of the IEEE **78**, pp.1464-1480, 1990.

[68] P. Ajjimarangsee, and T. Huntsberger, " Neural network model for fusion of visible and infrared sensor outputs," conference, SPIE VOL. 1003, sensor fusion: Spatial reasoning and scene interpretation, 1988.

[69] S. Blackman, and T. Broida, " Multiple sensor data association and fusion in aerospace applications," Journal of robotics system (3) , pp.445-485, 1990.

[70] E. Geraniotis, and Y. chau, "Robust data fusion for multisensor detection systems," IEEE Trans. on information theory, vol.36, Nov. 1990.

[71] A. Houles, and Y. Bar-shalon, " Multisensor tracking of a manoeuver target in clutter," IEEE Trans. on Aerospace and Electronic Systems, Vol AES-25, No.2, March 1989.

[72] C. Sword, N. Simaan, and E. Kamen, " Multiple target angle tracking using sensor array outputs ," IEEE trans. on aerospace and electronic systems, vol. 26, No. 2, March 1990.

[73] S. Grossberg, "Nonlinear neural networks: principles, machines, and architectures," Neural Networks, 1, pp.15-57, 1988.

[74] J.J. Hopfield, "Artificial neural networks," IEEE circuits and Devices Mag., pp.3-10, Sept. 1988.

[75] D.H. Ackley, G.E. Hinton, and T.J. Sejnowskii, " A Learning algorithm for Boltzmann machines," Cognitive Science, Vol.9, pp.147-169, 1985.

[76] J.S. Denker, "Neural networks models for learning and adaptation," Physica 22D, pp.216-232,1986.

[77] R. Hecht-Nielsen, " Theory of backpropagation neural networks," Proc. IJCNN-89, I-593, 1989.

[78] R.P. Lippmann, "Pattern classification using neural networks," IEEE Comm. mag., pp.47-64, Nov. 1989.

[79] E. Yair, and A. Gersho, "The Boltzmann Perceptron network: A soft classifier," Neural Networks, Vol.3, pp.203-221, 1990.

[80] S.I. Gallant,"Perceptron -based learning algorithms," IEEE Trans. on Neural Networks, Vol.1, No.2, pp.179-191, 1990.

[81] R. A. Iltis, and Pei-Yih Ting, " Computing Association Probabilities Using Parallel Boltzmann Machines," IEEE Transactions on Neural Networks, Vol. 4, No. 2, March 1993.

[82] I. Jouny, F.D. Garber, and S.C. Ahalt, "Classification of Radar Targets using Synthetic Neural Networks," IEEE Transactions on Aerospace and Electronic Systems, Vol.29, No.2, April 1993.

[83] James L. Crowley, and Yves Demazeau, " Principles and Techniques for Sensor Data Fusion," Signal Processing 32 (1993) pp.5-27 Elsevier.

[84] Huey-min Sun, and Shu-min Chiang, " Tracking Multitarget in Cluttered Environment," IEEE Transactions on Aerospace and Electronic Systems, Vol. 28, No.2, April 1992.

[85] Lawrence A. Klein, " A Boolean Algebra Approach to Multiple Sensor Voting Fusion, " IEEE Transactions on Aerospace and Electronic Systems, Vol. 29, No. 2, April 1993.

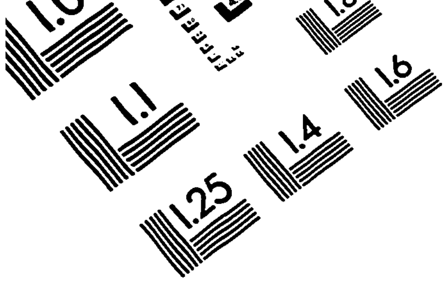
[86] Lang Hong, and Andrew Lynch, "Recursive Temporal Spatial Information Fusion with Applications to Target Identification," IEEE Transactions on Aerospace and Electronic Systems, Vol. 29, No. 2, April 1993.

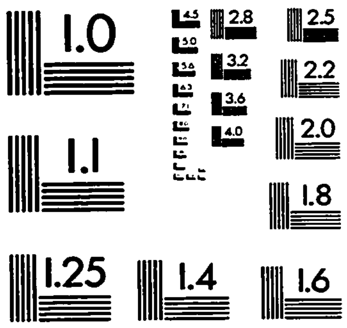
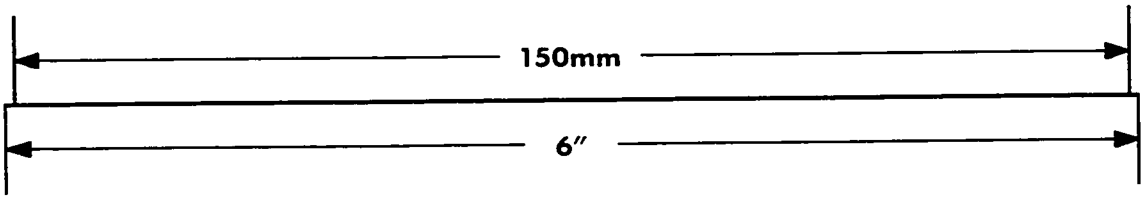
[87] Somnath Deb, Krishna R. Pattipati, and Yaakov Bar-shalom, " A Multisensor-Multitarget Data Association Algorithm for Heterogeneous sensors," IEEE Transactions on Aerospace and Electronic Systems, Vol. 29, No. 2, April 1993.

[88] Bart Kosko, **Neural Networks and Fussy Systems,** Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1992.

[89] Michael Chester, **Neural Networks,** Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1993.

[90] Lawrence A. Klein, **Sensors and Data Fusion Concepts and Applications,** SPIE–The International Society for Optical Engineering, Washington, USA, 1993.




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

150mm
6"