Neural Network Ensonification Emulation: Training and Application

Jae-Byung Jung

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2001

Program Authorized to Offer Degree: Electrical Engineering

University of Washington

Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Jae-Byung Jung

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

_____

Robert J. Marks II

Reading Committee:

_____

Mohamed A. El-Sharkawi

_____

Warren L. J. Fox

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Neural Network Ensonification Emulation: Training and Application

by Jae-Byung Jung

Chair of Supervisory Committee:

Professor Robert J. Marks II
Electrical Engineering

This dissertation investigates several modifications and extensions of conventional neural networks for application to the problem of optimally choosing the adjustable parameters in a sonar system. In general, neural networks offer several key advantages over other technologies that might be used for this task, including the ability to learn from examples and the ability to extract information about the underlying system through neural network inversion. One aspect of this work is the use of a neural network for emulating a computationally intensive acoustic model. A novel neural network training technique for varying output node dimension is developed, allowing a single neural network to be used for different output topologies. Step size modification for this training technique is also introduced to improve accuracy, convergence time, and the smoothness of the weight space, eventually providing better generalization. Inversion of neural networks is also investigated in order to solve for the optimal control parameters given a requested level of sonar performance. In order to improve inversion accuracy, modular neural networks are designed using adaptive resonance theory for pre-clustering. In addition, sensitivity of the feed forward layered

perceptron neural network is derived in this work. Sensitivity information (i.e., how small changes in input layer neurons affect output layer neurons) can be very useful in both the inversion process and system performance analysis. Finally, the multiple sonar ping optimization problem is addressed using an evolutionary computation algorithm applied to the results of properly trained neural networks. It searches for the combination of control parameters over multiple independent sonar pings that maximizes the combined sonar coverage.

TABLE OF CONTENTS

List of Figures iii

List of Tables vii

Chapter 1: Introduction 1

1.1 Adaptive sonar 1

1.2 Neural networks and evolutionary algorithms 2

1.3 Organization of dissertation 3

Chapter 2: Neural Network Training for Varying Output Node Dimension 5

2.1 Don't care training 6

2.2 Step size modification 7

2.3 Performance contrast: A sonar example 10

Chapter 3: Inversion of Neural Network 16

3.1 ART2: unsupervised training for clustering 18

3.2 Training comparison with a single network 20

3.3 Inversion of single neural network 22

3.4 Inversion of ART2 modular networks 28

Chapter 4: Neural Network Sensitivity Analysis 33

4.1 Neural network non-linear sensitivity 34

4.2 Sensitivity analysis for adaptive sonar neural network 37

Chapter 5: Team Optimization of Cooperating Systems 41
5.1 Preliminaries 41
5.2 Set covering problem 43
5.3 Area coverage with deformable shapes 45
5.4 Maximizing sonar area coverage 48

Chapter 6: Conclusions 57
6.1 Contributions 57
6.2 Ideas for future work 58

Bibliography 59

Appendix A: Sonar data 64
A.1 Sonar data 1 64
A.2 Sonar data 2 67

LIST OF FIGURES

2.1 The neural network training architecture. 7

2.2 Training performance comparisons with other training algorithms where pixels below the bathymetry are treated differently. 12

2.3 Test error for don't care training with and without SSM. Both cases used E_{AMSE} in Equation 4.8. The results of $\tilde{5}\%$, shown here, are typical for this problem. 13

2.4 Neural network performance comparison using a testing pattern 1. . . 14

2.5 Neural network performance comparison using a testing pattern 2. . . 15

3.1 Structure of ART2 Neural Network 19

3.2 Basic Architecture of ART2 Pre-clustering and Training 19

3.3 Comparison of RMS Errors. 21

3.4 Sample SE Maps of Testing. 23

3.5 Architecture of neural network inversion (shaded arrows represent clamped subsets while empty arrows involve iterative searching). 24

3.6 Neural network training and single output pixel inversion. 27

3.7 Multi Input Parameter Inversion and Maximizing Target Area. 29

3.8 Inversion of ART2 Modular Neural Networks (ANN 2 is selected when the Euclidian distance between the desired output and each cluster's centroid is measured in this example). 30

3.9 Multi input parameter inversion and maximizing target area using pre-clustered training by ART2. 31

4.1 Illustration of the neural network sensitivity analysis at the given operating point i_k with respect to the target surveillance output O_T. . . 34
4.2 Local gradient at the output node i. 36
4.3 Local gradient at the hidden node j. 36
4.4 Local gradient at the input node k. 37
4.5 Sensitivity analysis for adaptive sonar neural network at a given environmental situation 1 (the absolute sensitivities of 5 control input parameters and their output changes when each input was marginally increased by 2% of the full scale respectively as the rest 23 environmental parameters are fixed). 39
4.6 Sensitivity analysis for adaptive sonar neural network at a given environmental situation 2 (the absolute sensitivities of 5 control input parameters and their output changes when each input was marginally increased by 2% of the full scale respectively as the rest 23 environmental parameters are fixed). 40

5.1 Team optimization of cooperating systems (TOCS) for maximal area coverage (MAC). The K systems, in response to stimuli generate respective area coverages of $\{\vec{a}_k\}$. These coverages are combined and a fitness function is evaluated. The fitness provides input to the control which, in turn, generates changes in the control vectors, $\{\vec{c}_k\}$. 42
5.2 Illustration of the set covering problem. 43
5.3 Illustration of a translatable, rotatable rectangle with adaptive aspect ratio. 44
5.4 Coverage of an irregular shape with $K = 5$ rectangles of equal area. . 45

5.5 Snapshots of the process of coverage of a circle with $K = 2$ equal area rectangles. The entire circle cannot be covered. Each small block is representative of the coverage of circular target by 2 moving deformable boxes at the initial generation, 2nd generation, 15th generation, 20th generation, 35th generation, 50th generation, 587th generation, 630th generation, and the final generation, respectively, starting from upper left to lower right block. 47

5.6 Snapshots of MAC using TOCS for 4 boxes covering a circle. Each small figure is representative of the coverage of circular target by 4 moving deformable boxes at the initial generation, 11th generation, 19th generation, 24th generation, 58th generation, 109th generation, 243rd generation, 5628th generation, and the final generation, respectively, starting from upper left to lower right block. 48

5.7 MAC by TOCS as applied to sonar. The vector of environmental parameters, $\vec{e}$, is fixed. For the given environmental parameters, the combination of control vectors, $\{\vec{c}_k \mid 1 \leq k \leq K\}$, giving a combination maximal area of ensonification are desired. The control vector contains the parameters to be varied. The overall fitness value is equal to the area covered by the ensonification. A generic genetic algorithm is used to perform the search over the K vectors. 51

5.8 Population of genetic algorithm for sonar ping coverage optimization. 53

5.9 Fitness function convergence using different probabilities of crossover and mutation. 54

5.10 Modular Maximal Area Coverage Using a Neural Network Bank for Multiple Sonar Ping Problem (Best 4 different sonar pings contribute the maximal coverage after the convergence of GA fitness evaluations) 55

5.11 an experiment with SE map of sonar data 1. Best 2 sonar control sets are searched through the genetic algorithm to maximize the combined sonar coverage. 56

A.1 A sample SE map. 66

A.2 A sample TL map. 66

A.3 A sample SE map - simplfied version. 67

LIST OF TABLES

3.1 Training of ART2 pre-clustered data sets. 20
3.2 Training of a single neural network. 21

A.1 Environmental and control parameters used as inputs to train the neural network. 65
A.2 Environmental and control parameters used as inputs to train the neural network - simplified version. 67

ACKNOWLEDGMENTS

First, I would like to acknowledge that this work is supported by the Office of Naval Reserach.

I would like to express my sincere thanks and deepest appreciation to my advisor, Professor Robert J. Marks II, for his support and guidance throughout my studies and during the writing of this dissertation. This work would never have been completed without him.

I would also like to offer a special thanks to Professor Mohamed A. El-Sharkawi for the insightful comments and the solid guidance, without which this work would not be the same. Also, thank you to Professor Warren L. J. Fox for all the advice and helpful comments.

I would further like to offer a gratitude to my previous advisor, Il-Hong Suh, at Hanyang University in Korea for his thoughtful criticism and advice that helped build in-depth foundation of the course of my research.

I want to thank all of the members of Applied Physics Lab at the University of Washington, who were so generous in their support at various stages of this study. Especially, many thanks to Robert Miyamoto, Warren Fox, Greg Anderson, Chris Eggen, and Megan Hazen for their helpful comments and feedbacks along the way.

I wish to thank my colleagues of the Computational Intelligence Applications Laboratory at the University of Washington for sharing my memory and all the helpful feedbacks.

I cannot thank my parents enough for providing emotional, physical and mental nourishment throughout the years as well as the advice that helped set the course I am currently taking in life.

Finally, but most importantly, I would like to appreciate my wife, Kyung-Ja Shin, for her patience, understanding and encouragement throughout the long process.

Chapter 1

INTRODUCTION

1.1 Adaptive sonar

The objective of this dissertation is to explore sonar system adaptation and characterization through neural network training, inversion, and sensitivity analysis, and also to explore optimization techniques utilizing several forms of computational intelligence, including neural networks and evolutionary algorithms.

In underwater sonar surveillance, it is often required to determine a set (or subset) of sonar control parameters (such as the waveform to be transmitted or the depth at which to deploy the sonar) that maximize sonar performance over some part of the water column. In this work, the surveillance area considered is a two dimensional vertical slice of the ocean. Performance maps are calculated that show how the sonar will perform against hypothetical targets located at grid points within this vertical slice. The actual performance map is dependent on both sonar control and environmental parameters. A computationally intensive software model is typically used to emulate the acoustic propagation and scattering that go into determining the performance map.

For high fidelity acoustic models, however, it takes an enormously long time to generate even a single performance map, making the emulator not suitable for real time sonar control. One possible solution for this problem is the use of a neural network for a replacement of the intensive software model. There are, however, significant challenges in training neural networks in the presence of the huge dimensionality of

input-output relationship and varying bathymetry. This work introduces the development of a neural network training algorithm for this problem. The trained neural networks can then be used for inverting desired sonar performance to the optimal control parameters, and for determining sensitivity of performance to the various control and environmental parameters.

1.2 Neural networks and evolutionary algorithms

Neural networks offer several key advantages over conventional computing techniques that make them an excellent candidate for highly complex, nonlinear problems including the abilities as:

- learning from examples without a need for an explicit model of the problem,
- very fast execution time after training,
- performing well in a noisy environment,
- generalizing well,
- adapting to a changing environment via incremental training, and
- extracting useful information after training.

Evolutionary computation refers to a class of algorithms that seek to mimic the theory of biological evolution. Evolutionary algorithms operate on a population of potential solutions in parallel. By iteratively performing operations such as selection, mating and mutation, the population is altered with the intention of improving its overall fitness. An example of the most commonly used evolutionary algorithm is referred to as the genetic algorithm [5, 7, 13, 14, 16, 17, 30, 40]. Evolutionary algorithms are typically used in complex, high dimensional optimization problems. Evolutionary algorithms offer several advantages over traditional methods including:

- The ability to solve highly complex, nonlinear, discontinuous problems,
- The ability to handle complex constraints,
- The ability to scale well to high dimensional problems,
- They are relatively easy to understand and program.

1.3 Organization of dissertation

This dissertation is made up of 6 chapters. Chapters 2-4 provide don't care training and sonar performance adaptation and analysis through inversion and nonlinear sensitivity. Chapter 5 provides the maximal area coverage optimization problem using evolutionary algorithm. Finally, conclusion of this work is discussed in chapter 6. Additionally, sonar data sets regarding the training, inversion, and optimization in chapters 2-5 are given in the Appendix. The followings are brief descriptions of the contents of each of the chapters.

Chapter 2 of this dissertation introduces the development of the training technique for feed forward multi-layered perceptron neural networks, which allows a single neural network to train varying output node dimension. It includes don't care training with step size modification. Don't care training technique has irregularly selective weight update rule and step size modification compensates for the possible over training. An a posteriori probability that shows how often the weights associated with each output neuron are updated is obtained from the training data set and is used to evenly distribute the opportunity for weight update to each output neuron.

In addition to the training, the inversion of the trained neural network is presented in chapter 3. Gradient-based searching technique is applied to invert the trained neural network. ART2 unsupervised training is used for clustering in the output dimension so that several neural networks specializing in different output features are inverted using gradient descent technique.

Chapter 4 presents the neural network sensitivity analysis. It introduces two definitions of nonlinear function sensitivity including absolute and relative (logarithmic) sensitivity. Local gradients are accumulated by chain rule to get the change of surveillance output nodes with respect to the change of control parameters as the operating point or/and environment is provided.

In chapter 5, an evolutionary algorithm-based approach is applied to maximize the maximal area coverage described by the best combination of the finite number of systems, each of which provides local coverage map that needs not be the best coverage by itself. This modular multi-objective optimization problem accommodates a bank of trained neural networks. The computational intensity required for searching through multi dimensional solution space is significantly reduced by the architecture with the property of distributed modularity and genetic algorithms.

Chapter 6 presents conclusions and ideas for future work that are drawn from this work.

Chapter 2

NEURAL NETWORK TRAINING FOR VARYING OUTPUT NODE DIMENSION

Considered is the problem of neural network supervised learning when the number of output nodes can vary for differing training data. This work proposes irregular weight updates and learning rate adjustment to compensate for this variation. In order to accommodate for possible over training, an a posteriori probability that shows how often the weights associated with each output neuron are updated is obtained from the training data set and is used to evenly distribute the opportunity for weight update to each output neuron. The weight space becomes smoother and the generalization performance is significantly improved.

Multilayer perceptions (ML's) typically use a fixed network topology for all training patterns in a same data set. We consider the case where the dimension of the output can vary from training pattern to training pattern. Let the input-target output training data be $\{\vec{i}[n], \vec{t}[n] \mid 1 \leq n \leq N\}$. By different output dimensionality, we mean the dimensions of the output vector, $\vec{t}[n]$, can vary as a function of n. We assume the dimension, $M[n]$, of each output is known.

A modular neural networks structure [18][21][22] containing local experts for different dimension-specific training patterns can be applied to this problem. Here, each component neural network is trained with the subset of data corresponding to a fixed number of outputs. It becomes increasingly difficult, however, to implement a large number of neural networks for a large number of experts in the absence of ample training data. If a single neural network is to be used for the problem the output must be

made sufficiently large to handle the longest of target vectors. Let $M = \max_n M[n]$. We define a new output vector set of length M with elements, $\vec{\tau}[n]$, as the vector concatenation

$$\vec{\tau}[n] = \begin{bmatrix} \vec{t}[n] \\ \vec{t}_{DC}[n] \end{bmatrix} \tag{2.1}$$

where the vector $\vec{t}_{DC}[n]$ (the DC is for don't care), of length $M - M[n]$, corresponds to output values not in the active region. If the n^{th} output is partitioned into Active_n and $\mathsf{Don't\ Care}_n$, then (2.1) can equivalently be written as

$$\begin{aligned} \tau_m[n] &= (\vec{\tau}[n])_m \\ &= \begin{cases} t_m[n], & m \in \mathsf{Active}_n \\ \left(\vec{t}_{DC}[n]\right)_m, & m \in \mathsf{Don't\ Care}_n \end{cases} \end{aligned} \tag{2.2}$$

One possible approach to use a single neural network is filling $\vec{t}_{DC}[n]$ with arbitrary constant values, and then training it as if it had a fixed output node dimension using conventional training. The problem, however, is that undesirously abrupt change between $\vec{t}[n]$ and $\vec{t}_{DC}[n]$. This artifact makes the neural network do unnecessary effort during the training so the overall performance is not expected so good. Another alternate approach is replacing $\vec{t}_{DC}[n]$ with their adjoining neighborhood in $\vec{t}[n]$ by extrapolation. Although this smearing method makes more sense than the first approach, the output dimension is still unnecessarily stretched out to fix the output dimension. Therefore, a novel approach is required.

2.1 Don't care training

For don't care training, the partition of the output, varying for each training data pair, is that in (2.1). The input representation is augmented. The first component, $\vec{i}[n]$, of the augmentation contains the conventional input training data. The second

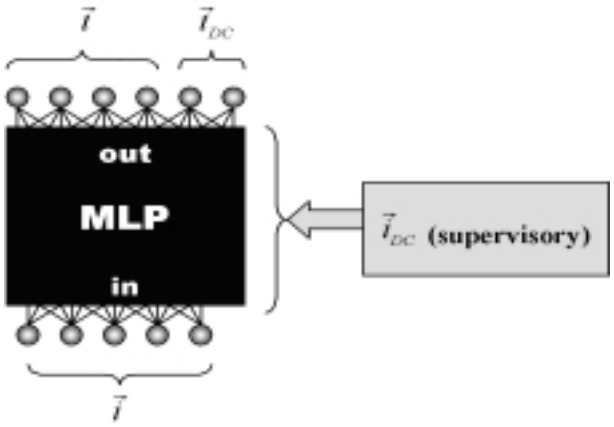

Figure 2.1: The neural network training architecture.

component, $\vec{i}_{DC}[n]$, dubbed the don't care input, contains characterization and statistics of those output values with a "don't care" status. The assignment of each output neuron to either $\vec{t}_{DC}[n]$ or $\vec{t}[n]$, for example, is determined by $\vec{i}_{DC}[n]$. The dimensions of $\vec{t}[n]$ and $\vec{t}_{DC}[n]$ vary with n as dictated by $\vec{i}_{DC}[n]$. The dimensions of $\vec{i}_{DC}[n]$ and $\vec{i}[n]$ are static. The don't care input is not used as conventional neural network input data but, rather, is used in

- training to alter learning parameters as a function of n, and
- testing to specify which of the output neurons have don't care values and therefore should be ignored.

2.2 Step size modification

During error back-propagation training, the weights connected to the don't care output neurons are not updated while other weights are updated with a modified step size. An empirical a posteriori probability showing how often the weights associated with each output neuron are updated is obtained from the training data set and

is used to give the even amount of opportunity for weight update to every output neuron. The empirical probability of weight update associated with the mth output neuron is defined as the ratio of the frequency of weight update associated with the output neuron m, denoted by $f_o[m]$, to the total number of training patterns, N.

$$p_o[m] = \frac{f_o[m]}{N}. \tag{2.3}$$

It is reasonable to give a large gain to the weight that has less opportunity of correction. Thus, the step size modification (SSM) is defined by

$$\begin{cases} \eta_m^o = \frac{\eta}{p_o[m]}, & \text{for active output neurons;} \\ \eta_m^o = 0, & \text{for don't care output neurons;} \\ \eta^h = \eta, & \text{for all other neurons.} \end{cases} \tag{2.4}$$

where η is a global learning rate used for the weight update of ordinary error back propagation, η_m^o is the modified learning rate for the weights of output neuron m and η^h is the step size for all other weights.

The difference between the target output value, $t[n]$, and the actual output, $o[n]$, of neuron n included in $o[n]$ at the n^{th} training pattern vector is used to take the instantaneous sum of squared errors of the network.

A commonly used variation is batch-mode learning. The nth pattern is evaluated to obtain the derivative terms, $\frac{\partial E(n)}{\partial w}$, which are summed to obtain the total derivative,

$$\frac{\partial E}{\partial w} = \sum_{n=1}^{N} \frac{\partial E(n)}{\partial w},$$

used in batch mode training. For don't care training, the sum is

$$\frac{\partial E}{\partial w} = \sum_{t_m[n] \in \text{Active}_n} \frac{\partial E(n)}{\partial w}.$$

The step size is modified to

$$\begin{cases} \hat{\eta}_m^o &= \frac{\hat{\eta}}{p_o[m]}, & \text{for every output neuron } m; \\ \hat{\eta}_m^h &= \hat{\eta}, & \text{otherwise.} \end{cases} \tag{2.5}$$

where $\hat{\eta}$ is a global learning rate used for the batch-mode weight update of ordinary error back propagation, $\hat{\eta}_m^o$ is the modified learning rate for the weights of output neuron m and $\hat{\eta}^h$ is the step size for all other weights.

The mean squared error (MSE) is obtained by summing $E[n]$ over all n and then normalized with respect to the set size N [5],[6]. Specifically

$$E(n) = \frac{1}{2} \sum_{m \in \text{Active}} (t_m[n] - o_m[n])^2 , \tag{2.6}$$

and

$$E_{\text{MSE}} = \frac{1}{N} \sum_{n=1}^{N} E(n). \tag{2.7}$$

The MSE, however, is an inappropriate representaion of training error for variable output dimensionality. For don't care training, rather, the average mean squared error (AMSE), representing the mean squared error per each output element, is more appropriate.

$$E_{\text{AMSE}} = \frac{1}{N} \sum_{n=1}^{N} \frac{E(n)}{f_o[n]}. \tag{2.8}$$

An output neuron, even when sparsely used, is represented equally in the composite error totally.

2.3 Performance contrast: A sonar example

Sonar data corresponding to various environmental and sonar control parameters was generated from an computationally intensive acoustic model. The ensonification[1] map is arranged in a 75 (range) $\times$ 20 (depth) pixel image[2]. There are therefore (a maximum of) $1500 = 75 \times 20$ values for each training data pair output. The bathymetry [3] is one of the environmental parameters varied. When a neural network is trained with the ensonification map as output, neurons assigned to pixels lying below the ocean bottom cannot be ensonified and are therefore classified as don't care output neurons. Twenty eight environmental and control parameters, detailed in Table A.1, were used as inputs. A total of 5000 input-output profiles were used in the training of the neural network and 3000 were reserved for testing.

The comparison of typical training performance with the algorithms using fixed arbitrary numbers and smearing method as stated earlier is shown in Figure 2.2. The top curve corresponds to fixing the don't care pixels to arbitrary values. The next plot, corresponding to a sequence of dots, results from replacing column of pixels under the bathymetry equal to the deepest pixel value in the column. We did this procedure as smearing. The second plot from the bottom, shown as a broken line, is for don't care training without SSM. The bottom solid plot is don't care training with SSM. Don't care training invariably outperforms the other algorithms in terms of training error as well as convergence time.

Figure 2.3 illustrates the most important characteristics of don't care training. When SSM is not used in don't care training, the testing error starts to increase at 5000 epochs whereas the other testing error curve with SSM keeps decreasing.

[1] As measured by the acoustic signal excess or transmission loss. More details about the acoustic emulation are in Jensen et.al.[8].

[2] The range is from 0 to 15 km. The depth is from 0 at the surface of the water to 400 m. Sampling is uniform. See Appendix A for details of sonar data 1.

[3] The shape of the ocean's floor. The science of sounding or measuring depths in the ocean.

Therefore, it is obvious that the generalization capability is improved by applying SSM to don't care training. The testing error curves in Figure 2.3 are very typical for this problem.

Figures 2.4 and 2.5 shows testing examples where the neural networks and their absolute errors are compared with the desired values. Points below the bathymetry correspond to unspecified output nodes ascribed a don't care status for this specific pattern. The plot labeled Target is desired ensonification map, NNA is the neural network trained using arbitrary fixed numbers, NNB is the neural network trained smearing, and NNC is the neural network trained by the SSM training technique and the ASME error in Equation 4.8. The output nodes adjoining unspecified region have big errors in NNA, whereas NNB improved this problem significantly. However, NNB still has a big error on the rest of the region. If we take a look at maps on the right hand side column illustrating absolute differences, it is obvious that NNC, in terms of final accuracy, outperforms the other two techniques.

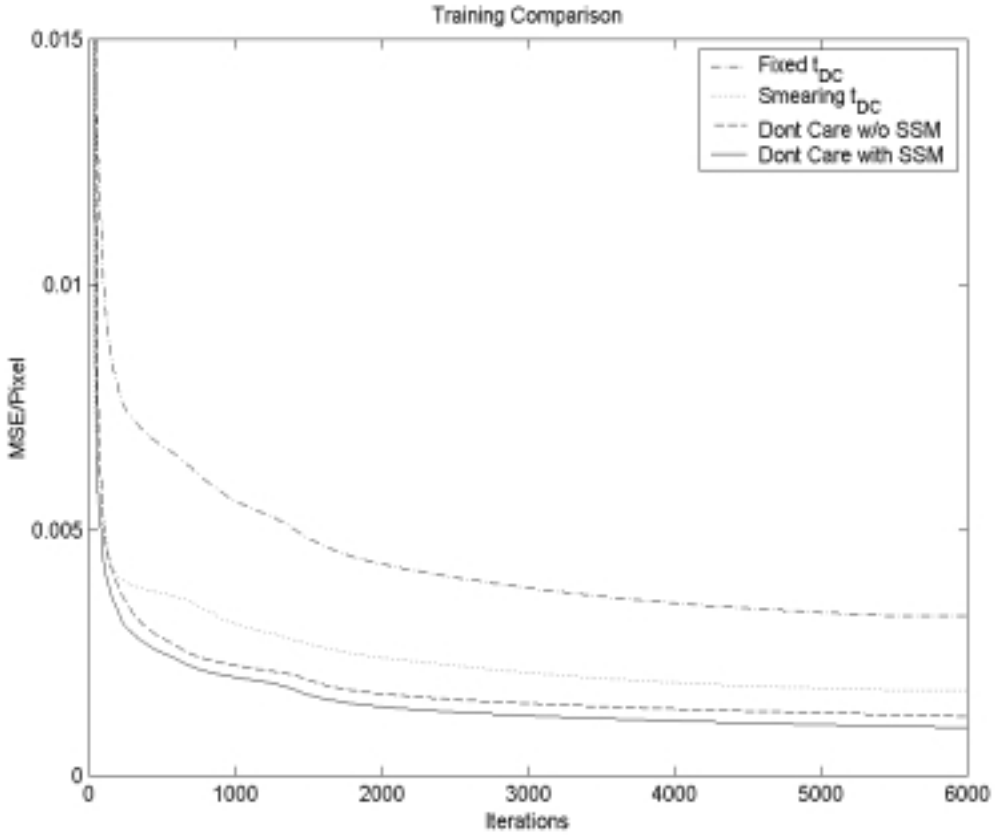

Figure 2.2: Training performance comparisons with other training algorithms where pixels below the bathymetry are treated differently.

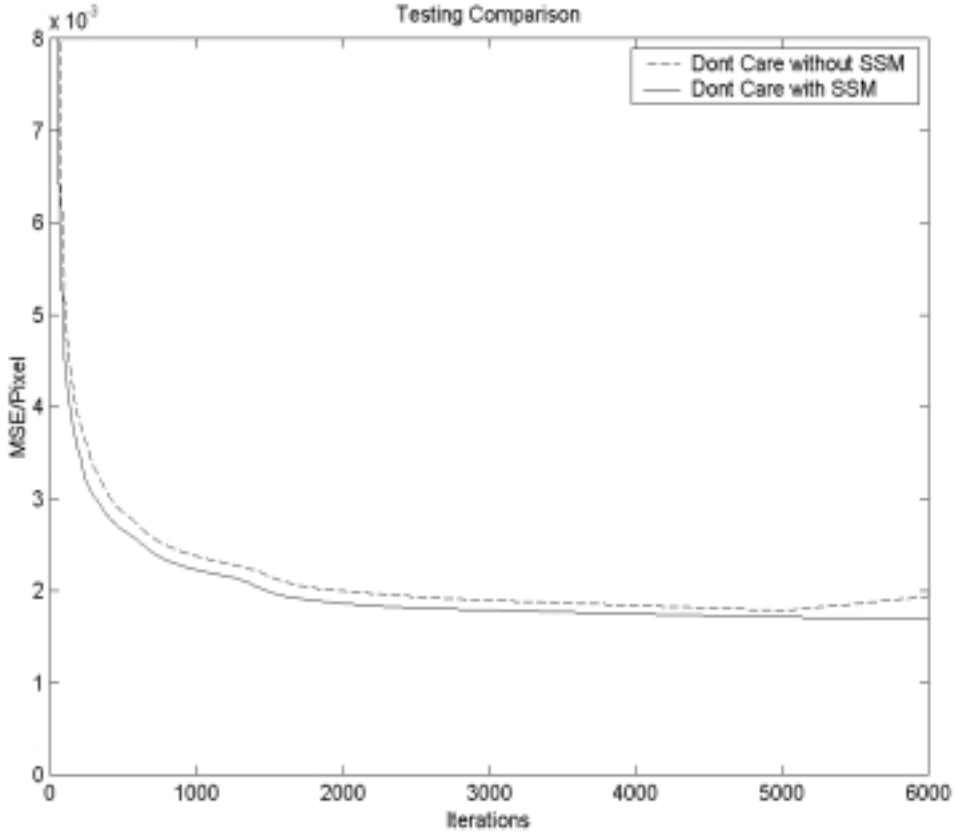

Figure 2.3: Test error for don't care training with and without SSM. Both cases used E_{AMSE} in Equation 4.8. The results of $\tilde{5}\%$, shown here, are typical for this problem.

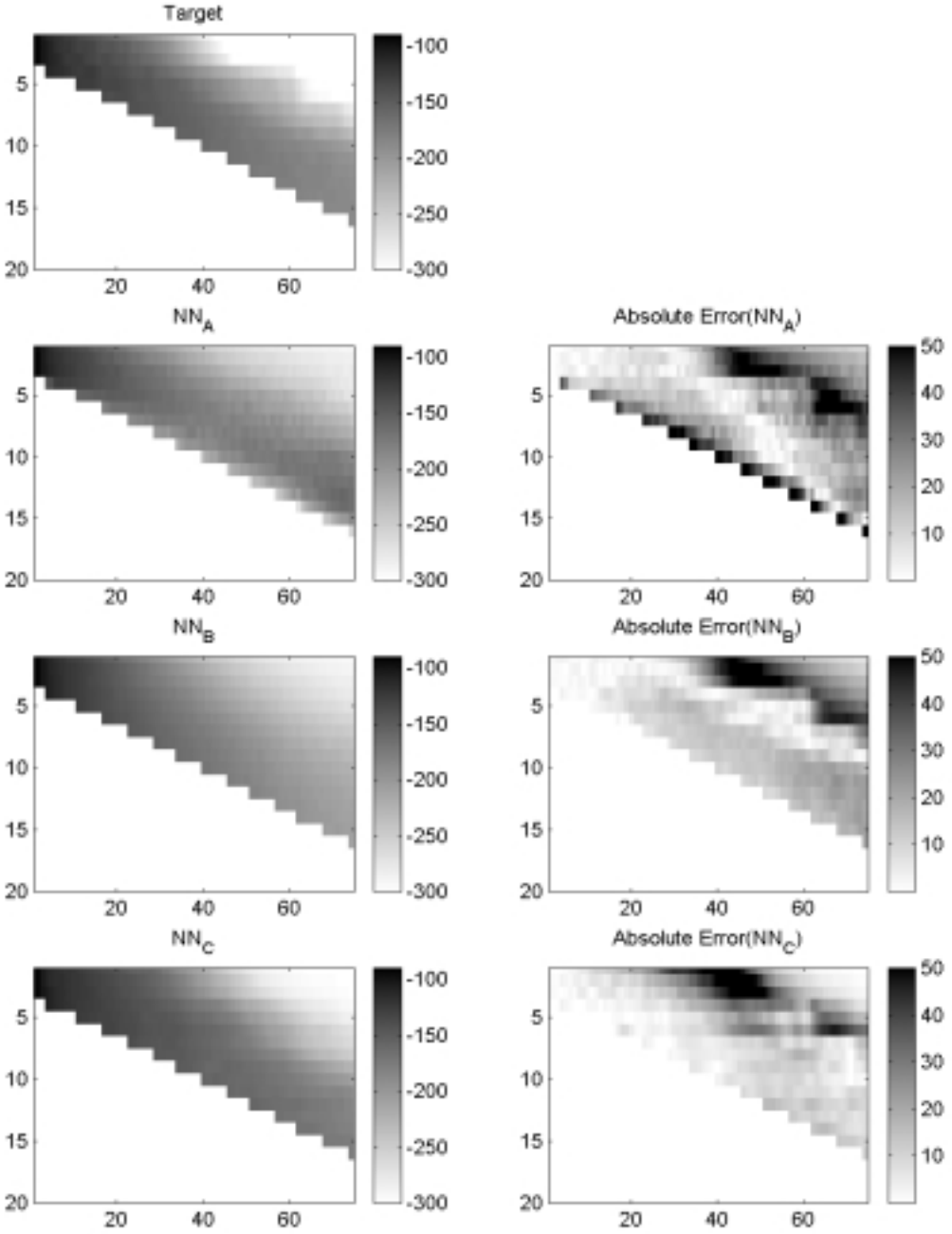

Figure 2.4: Neural network performance comparison using a testing pattern 1.

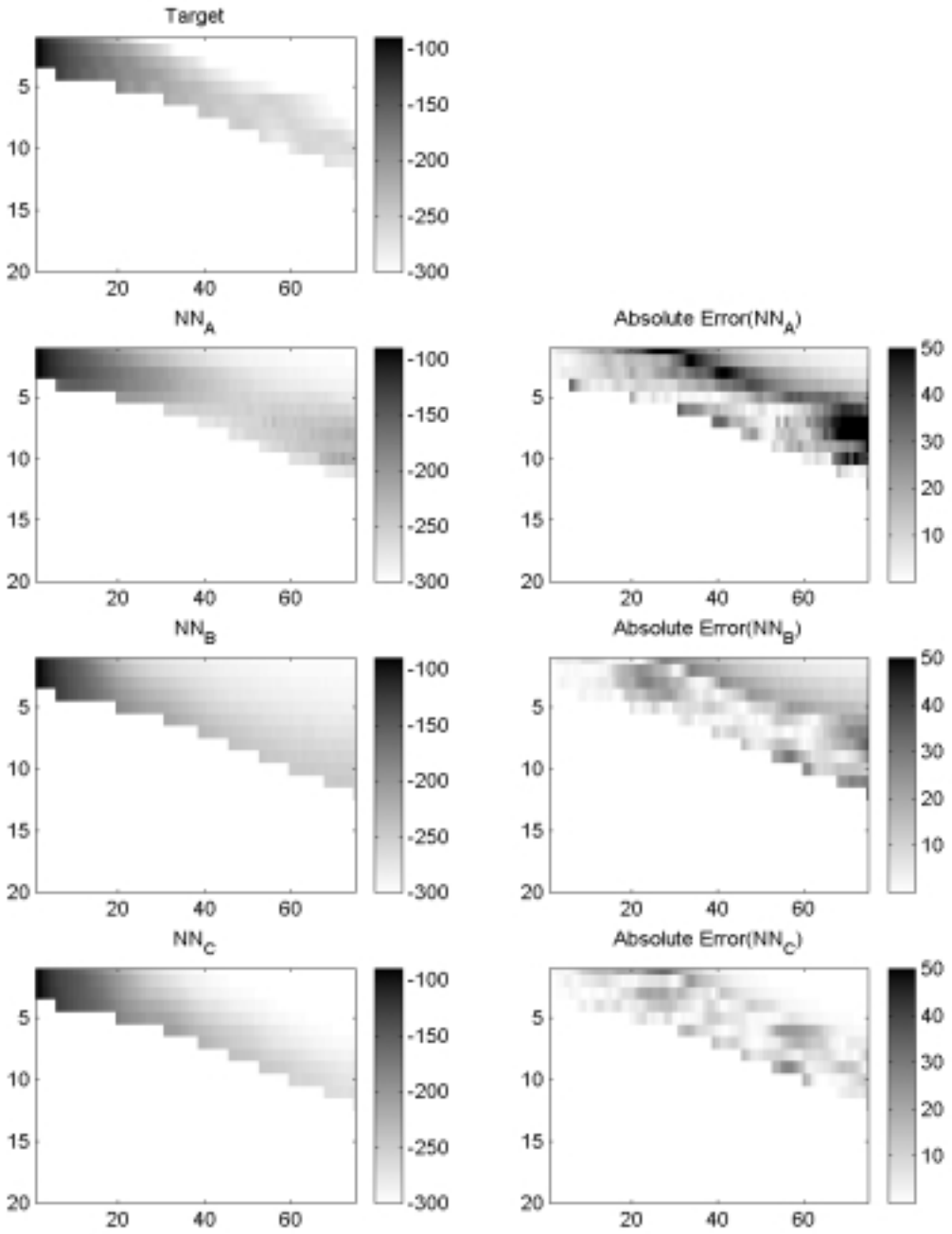

Figure 2.5: Neural network performance comparison using a testing pattern 2.

Chapter 3

INVERSION OF NEURAL NETWORK

Feedforward layered perceptron neural networks seek to capture a system mapping inferred by training data. A properly trained neural network is not only capable of mimicking the process responsible for generating the training data, but the inverse process as well. Neural network inversion procedures seek to find one or more input values that produce a desired output response for a fixed set of synaptic weights. There are numerous methods for performing neural network inversion. Multi-element evolutionary inversion procedures are capable of finding numerous inversion points simultaneously [8]. Constrained neural network inversion requires that the inversion solution belong to one or more specified constraint sets. In many cases, iterating between the neural network inversion solution and the constraint set can successfully solve constrained inversion problems. Neural network inversion is illustrated by its use as a tool in query based learning and sonar performance analysis

Inversion of a system that produces appropriate inputs from the given desired target outputs is useful in many areas. The notion of single-element network inversion via the gradient approach was first proposed by Williams [41] and later by Linden and Kinderman [28][29]. Their idea is based on the standard error back-propagation optimization. As is of the case with many gradient technique, gradient descent error back propagation is a deterministic algorithm used for energy minimization suffering from a fundamental weakness : it may get stuck in one of the local minima that are not globally optimum. But, in many important cases, the selection of different initial points and/or heuristically adaptive step size usually solve the problem in practice.

Neural network inversion is highly dependent on its training performance because

the inversion process needs to use the trained neural network whose weights are clamped after the network is fully trained. From the iterative inversion process, the optimal input vector is obtained to satisfy the specified output performance.

When a single training data is divided into smaller data clusters by appropriate similarity measurement, each cluster can have different characteristics from the others. Hence, the training of each individual neural network performs better than a single neural network trained by the entire data set, because each data cluster is composed of its unique pattern vectors. Thus, the corresponding neural network learns only similar input-output relationship within the same cluster.

The choice of suitable classifier is very important, but highly problem dependent. Basically, every classification technique falls into two types based on the existence of a priori information of the number of clusters. For examples, the K-nearest neighbor method needs specific a priori information about the actual data class, and SOFM (Self-Organizing Feature Map) [27] and ART [4][3] are useful when no information about the actual class is physically available.

Besides, the mixture of experts by Jordan and Jacob [21][22] employs a set of expert networks and gating network structure. The gating network modulates the outputs of the expert networks to produce a single modular network output. Both the gating network and expert networks are typically single layer linear perceptrons since simple models are desired for localized fits. The mixture of experts can be viewed as modeling the conditional probability of the desired target output given the network input from the training data set, and this a posteriori probability is back-propagated to adjust the weights of both the expert networks and gating network. hierarchical mixture of experts was proposed to solve nonlinear supervised learning problems [22]. The architecture is a tree in which the gating networks sit at the nonterminals of the tree, while the expert networks sit at the leaves of tree. The output vectors from each expert network proceed up the tree, being blended by the gating network outputs. However, the computational cost increases as the spawning

of the branches from the tree so that more gating and expert networks need to be trained. For the sonar acoustics analysis problem, pre-clustering of the sonar data set in the output space using ART2 network and its neural network inversion are shown to work better than a single neural network.

3.1 ART2: unsupervised training for clustering

ART is widely used unsupervised learning network developed by Carpenter and Grossberg in 1987 [4][3]. While ART1 is designed for clustering binary vectors, ART2 accepts continuous-valued vectors. With ART, the number of clusters is automatically and adaptively selected using a vigilance parameter that allows the users to control the degree of similarity of patterns placed on the same cluster.

Figure 3.1 illustrates the basic structure of ART2 neural network composed of three layers. The $F1$ layer is an input processing field comprising the input portion and the interface portion. The $F2$ layer is a cluster unit that is a competitive layer in that the units compete in a winner-take-all mode for the right to learn each input pattern. The third layer is a reset mechanism that controls the degree of similarity of patterns placed on the same cluster.

In the sonar acoustic data clustering of sonar data 2[1], every 390 dimensional output SE vector is provided to the $F1$ layer of ART2 network and unsupervised ART2 learning is performed to classify the entire data set into smaller pieces of sub sets. This process is depicted in Figure 3.2. The SE maps of the entire data set are, therefore, split into 5 smaller clusters in the output space using the vigilance parameter.

[1]See Appendix for details of input output relationship

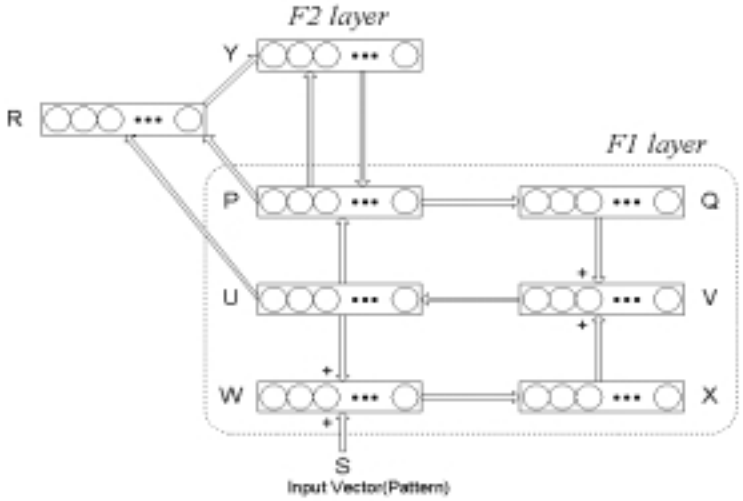

Figure 3.1: Structure of ART2 Neural Network

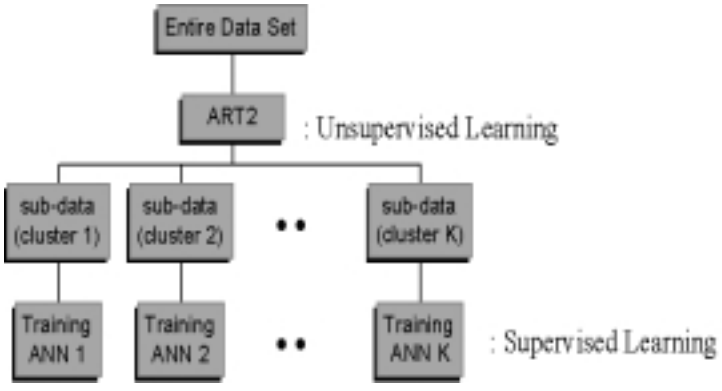

Figure 3.2: Basic Architecture of ART2 Pre-clustering and Training

Table 3.1: Training of ART2 pre-clustered data sets.

Cluster ID	Number of Patterns	Neural Network Topology	Training Epochs	Average RMS Error	Training Algorithm
1	499	5-10-30-390	100,000	1.95625	Quick Prop
2	585	5-10-30-390	100,000	1.57272	Quick Prop
3	409	5-10-30-390	100,000	1.69862	Quick Prop
4	286	5-10-30-390	100,000	1.58875	Quick Prop
5	221	5-10-30-390	100,000	1.89689	Quick Prop

3.2 Training comparison with a single network

When the entire data set is used to train a single network without pre-clustering, a bigger neural network structure is needed. In our application, the structure of 3 hidden layers is found to be the best in terms of training and testing errors. With the same training parameters and training algorithm, the network training error for single network is worse than those for ART2 pre-clustered neural networks. Furthermore, with simpler network topology, ART2 pre-clustered neural networks achieved better training results as illustrated in table 3.1 and 3.2. The results are shown in Figure 3.3 which illustrates the final RMS error of the training.

To test all the neural networks, 500 testing patterns not used by either neural networks for trainings are applied to the networks. Figure 3.4 shows some examples of testing results randomly picked from the testing data set. The values of six patterns are color coded. The cluster identifications appear on the left most labels on the left side of the first column. The desired target patterns are in the first column. The actual testing output patterns from the single network without pre-clustering are

Table 3.2: Training of a single neural network.

Number of Patterns	Neural Network Topology	Training Epochs	Average RMS Error	Training Algorithm
2,000	5-12-12-30-390	100,000	2.19659	Quick Prop

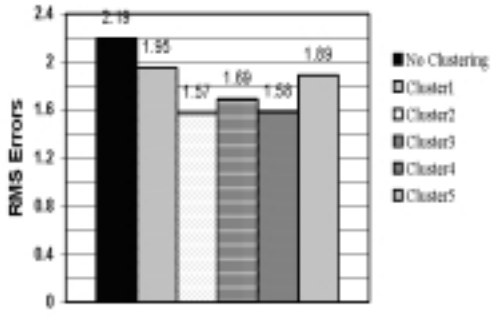

Figure 3.3: Comparison of RMS Errors.

shown in the second column. The third column has the absolute differences between the desired target patterns and the testing patterns. The fourth column is for the testing of ART2 neural networks. The last column includes the absolute differences between the desired target output maps in the first column and the ART2 testing output maps in the fourth column.

The testing performance of the single network is generally good, but it can miss drastic changes or strong stripes in SE map. Especially, the first row of Figure 3.4 depicts distinct features and shows improvement compared with single neural network. Single network didn't catch the changing shadow area that has very low SE values on the right part of SE map, while the ART2 pre-clustered network that fell into the fourth cluster achieved much better improvement in such locally concentrated error beside the numerical achievement in average RMS error. This observation is consistent with several other test patterns.

3.3 Inversion of single neural network

A neural network approach to the inversion is illustrated in Figure 3.5. Once the neural network is trained, the inverted neural network can provide the control input parameter that reproduces the desired SE values in a specified surveillance region. In figure 3.5, the environmental parameters such as wind speed, surface sound speed, bottom sound speed, and bottom type are assumed fixed and are therefore clamped to specific values. The target region is a subset of SE map. The remaining region is in the don't care category and is allowed to float in the inversion process to arbitrary values typically constrained to lie within a specified range. The network is inverted to specify the best sonar settings for a fixed set of environmental parameters using iterative gradient inversion technique. After the optimal sonar parameter was obtained, the feed forward neural network is used to evaluate the inversion accuracy.

The gradient approaches for the inversion of multilayer perceptrons make use

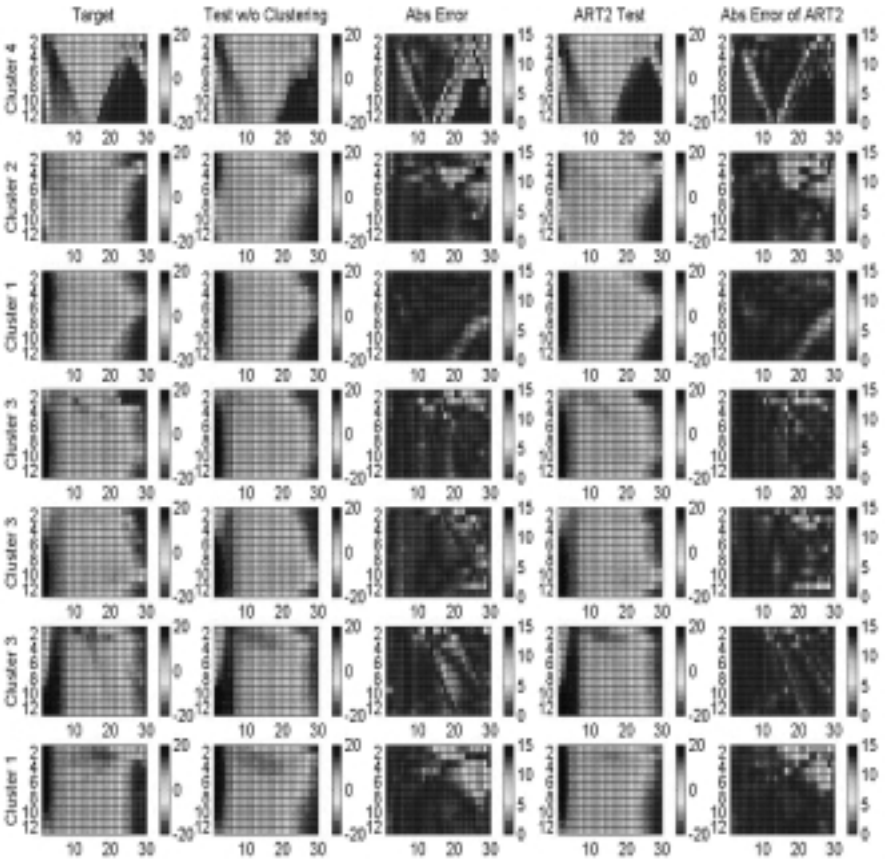

Figure 3.4: Sample SE Maps of Testing.

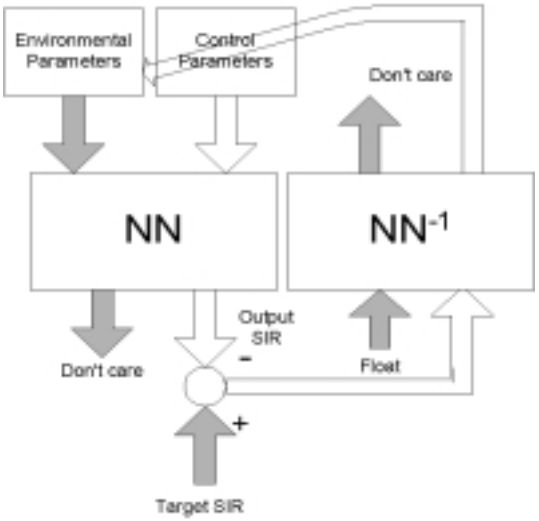

Figure 3.5: Architecture of neural network inversion (shaded arrows represent clamped subsets while empty arrows involve iterative searching).

of the well known back-propagation algorithm [12][20][34][37][39][42]. The inversion performance highly depends on the initial input vector. In practice, if the inversion error is not within small range and is stuck in a local minima, a new initial input vector is generated until the desired performance is achieved. When we want to find a subset of input vector, i, so that minimize the objective function, $E(i)$, which can be denoted as follows.

$$E(i) = \frac{1}{2}(o_i - t_i)^2 \tag{3.1}$$

where o_i is the neural network output for input I, and t_i is the desired output for input i. The search is initialized with an input vector i^0. If i_k^t is the k^{th} component of the vector $\vec{i}^t$, then gradient descent suggests the recursion

$$i_k^{t+1} = i_k^t - \eta \frac{\partial E}{\partial i_k^t} \tag{3.2}$$

where η is the step size and t is the iteration index. Assuming a general feed-forward topology, the iteration for inversion in the equation above can be solved as follows,

$$\frac{\partial E}{\partial i_k} = \delta_k, k \in I \tag{3.3}$$

Therefore, the gradient information for any neuron j is,

$$\delta_j = \begin{cases} \varphi'(net_j)(o_j - t_j), & j \in O \\ \varphi'(net_j) \sum_{m \in H,O} \delta_j w_{jm}, & j \in I, H. \end{cases} \tag{3.4}$$

where,

I, H, O are the sets of input, hidden, and output neurons respectively,
w_{jm} is the weight value connecting the neuron j to the neuron m,
φ' is the derivative of the j^{th} neuron squashing function,
o_j is the activation of the j^{th} neuron,
t_j is the desired output of the j^{th} neuron, and
net_j is the weight sum of the incoming signal to the j^{th} neuron.

Note the neuron derivatives, δ_j, must be solved in a backward order from output to input similar to the standard back-propagation algorithm. The inversion performance is highly dependent on the initial input vector i^0. In practice, if the inversion error is not within small range and looks like being stuck in a local minimum, a new initial input vector should be generated until the desired performance is achieved.

3.3.1 Single element inversion

The subset of outputs to be inverted is confined in one output pixel at a time during an inversion process while other outputs are floated[2]. The network can be inverted to find the best sonar depth to achieve the inversion target SE value that is given as a clamped value. To illustrate, four environmental parameters are clamped (fixed) to specific values of wind speed = 7m/s, sound speed at surface = 1500m/s, sound speed at bottom = 1500m/s, and bottom type = 9(soft mud). The optimum input control parameter is achieved during the iterative inversion process. This optimum input is injected into the feed forward neural network to reproduce a SE value that can be compared with the target SE. Therefore, 390 individual pixel inversions are performed across the entire output surveillance area, and the reproduced output SE values are put together in a SE map.

The comparison of the inversion result with the training result is illustrated in Figure 3.6. Three plots in the left column represent training target SE map, neural network output map, and absolute errors between two SE maps, respectively. Inversion target SE map, inversion result after 390 individual inversions, and absolute errors between two SE maps are shown in the right column respectively. In most cases, the inversion process provided the best sonar depth, and the sonar depth coupled with clamped environment reproduced the desired SE values.

3.3.2 Multiple parameter inversion and maximizing the target area.

In general, the subset of input parameters to be inverted need not be confined only a single input parameter. Besides, multiple output SE values can be inverted at a time. In this experiment, the input parameter set includes wind speed, sound speed at surface, sound speed at bottom, bottom type and sonar depth. Values of these parameters are inverted. The output target area is tiled with 2x2 pixel regions. Thus,

[2]SE maps of sonar data 2 are used in this experiment. See Appendix A for details.

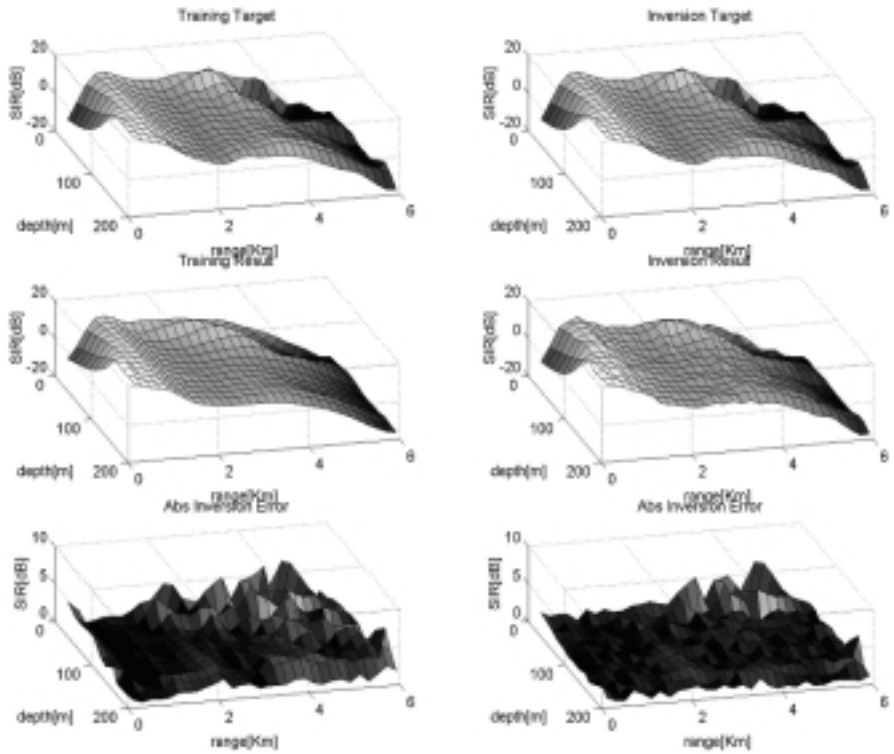

Figure 3.6: Neural network training and single output pixel inversion.

2x2 output pixel groups are inverted one at a time to find out the best combination of these 5 input parameters to satisfy the corresponding SE values. This 2x2 kernel window moves after each inversion until the entire SE map is covered without overlap.

Practically, for a given target area, the exact SE values to which to be inverted are not known. Rather, the maximum deliverable SE is desired. This can be achieved by setting the output region of interest to values that are the maximum achievable SE values as determined by the training data set. In such a case, the objective function can not always reach zero, because the training was not perfect. This limitation obtains the maximum values given by the training data set. The minimum value reached by the error function, however, corresponds to inverted input values that produce the maximum SE in the output target region.

Three plots on the left column in Figure 3.7 show (1) the inversion target whose pixels are the maximum achievable SE values, (2) inversion result reproduced from both four inverted input parameters and a clamped input parameter, and (3) the absolute error between two maps, respectively. The other column of plots shows the normalized representation of inversion results when each pixel is normalized from 0 to 1. The actual inversion error appearing in the lower left SE map of Figure 3.7 is caused primarily by the imperfect training of a huge single neural network. However, it is obvious that the better the training is the less the inversion error. Inversion of ART2 modular neural networks that have better training performance is discussed in the next section.

3.4 Inversion of ART2 modular networks

Because the inversion process makes use of a trained neural network, it is obvious that the better training performance, the less inversion error. The limitation to reach the maximum value using a single neural network can be improved when we employ the inversion of ART2 modular neural networks. Suppose that the original pre-clustering

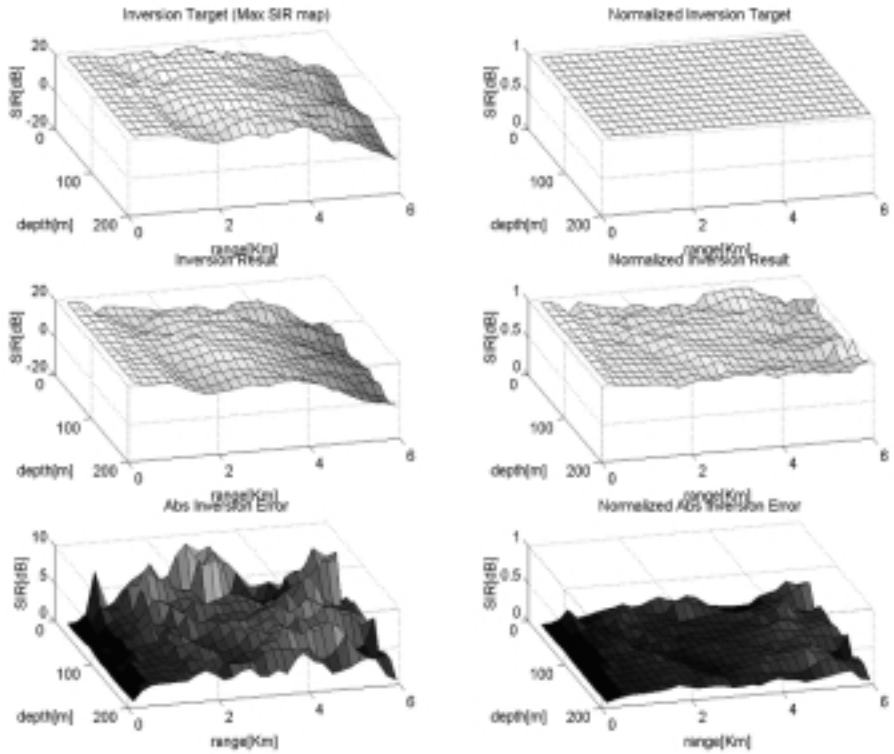

Figure 3.7: Multi Input Parameter Inversion and Maximizing Target Area.

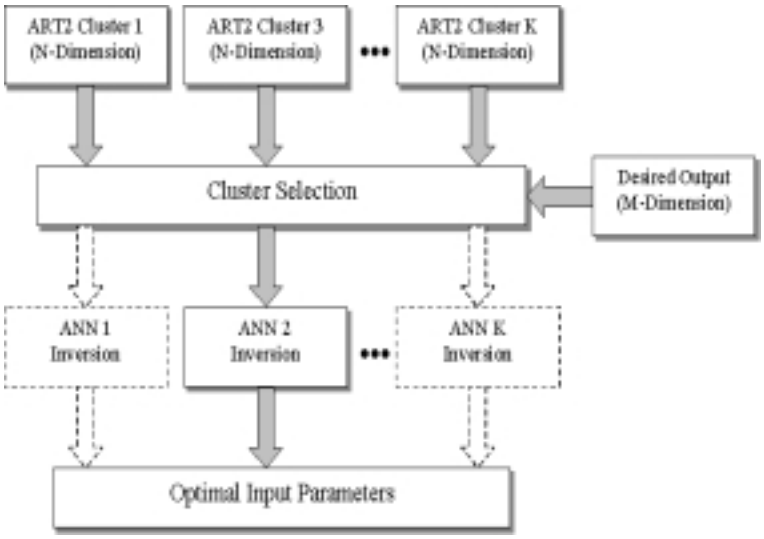

Figure 3.8: Inversion of ART2 Modular Neural Networks (ANN 2 is selected when the Euclidian distance between the desired output and each cluster's centroid is measured in this example).

was done in N dimensional hyper space and we have $M\,(\leq N)$ dimensional desired output vector to be clustered. Then, N dimensional cluster centroids can be projected onto M dimensional sub space, and the distance from the desired output vector to each cluster centroid is measured as a metric. Each cluster has its own centroid. Euclidian distance between a given vector and each projected centroid can be used as a metric for computational simplification to select the inversion target network as illustrated in Figure 3.8.

Figure 3.9 shows the improved inversion result by pre-clustering. Every inversion scheme is the same as that used in the previous section: which 4 pixels are maximized by sliding a 2x2 kernel window through the entire SE map in order to get the best combination of 5 input parameters. According to the location of the 2x2 kernel window, a neural network is selected by the Euclidian metric for inversion, and it moves after each inversion until the entire SE map is covered without overlap.

Three plots in the left column of Figure 3.9 show (1) the inversion target whose

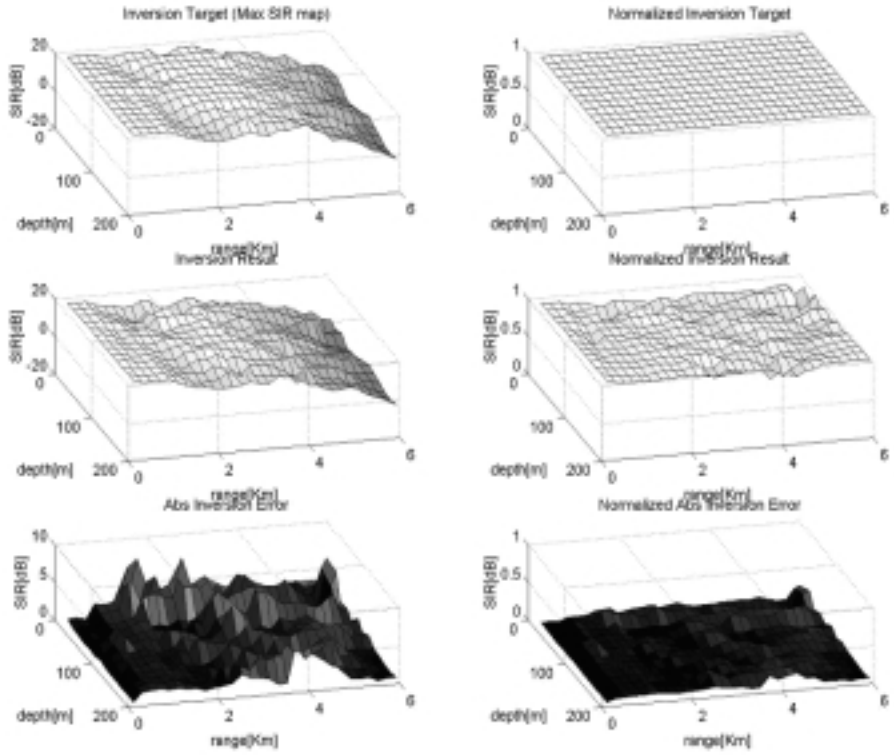

Figure 3.9: Multi input parameter inversion and maximizing target area using pre-clustered training by ART2.

pixels are the maximum achievable SE values, (2) inversion result which is reproduced from both four inverted input parameters and a clamped input parameter, and (3) the absolute error between two maps, respectively. The other column of plots includes the normalized representation of inversion results when each pixel is normalized from 0 to 1. The actual error in the lower left map of Figure 3.9 is less than that of Figure 3.7, due to the optimal selection of better neural network.

Chapter 4

NEURAL NETWORK SENSITIVITY ANALYSIS

The sensitivity of neural network is investigated in this chapter. Sensitivity analysis is very useful, especially, for the following issues.

- Sensitivity analysis can be used for feature selection as neural network is being trained or after the training. Also, it is found useful to eliminate superfluous input parameters, thereby reducing the dimension of the decision space and increasing speed and accuracy of the system [2].

- It is possible to develop a neural network that can ideally be trained to perform high precision computation. However, when implemented in hardware, the non-linearity occuring in the operation of various network component may practically make a network impossible to train significantly, depending on the overall system architecture. The sensitivity analysis procedure is very important in the investigation of these non-ideal effects. In other words, the sensitivity analysis of neural network is an important issue from the view point of engineering [23].

- Once neural network is trained, it is very important to determine which of the control parameters are critical to the decision making at a certain operating point such that environmental situation or/and control constraints are provided. This can be done through input parameter sensitivity analysis.

Above all, the last issue of the sensitivity analysis will be mostly studied in this chapter.

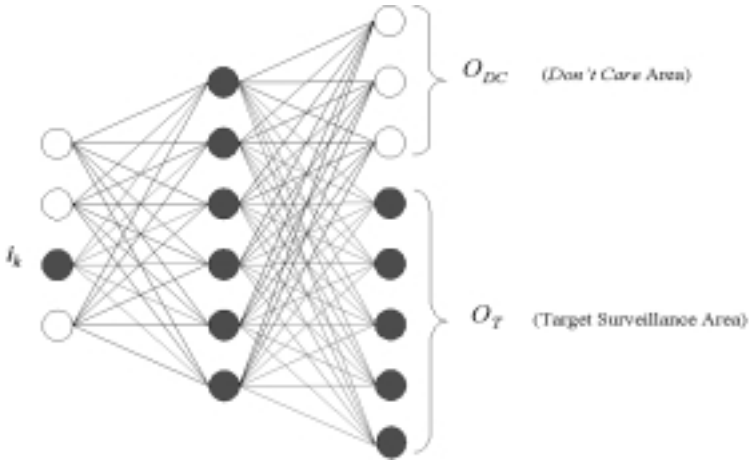

Figure 4.1: Illustration of the neural network sensitivity analysis at the given operating point i_k with respect to the target surveillance output O_T.

4.1 Neural network non-linear sensitivity

To obtain the sensitivity of the neural network with respect to input parameter I, let's define the absolute sensitivity of non-linear function, f, as

$$S = \frac{\partial}{\partial I} f(I) = \frac{\partial O}{\partial I}. \tag{4.1}$$

where I is a set of the input parameters and O is a set of the output parameters of the neural network.

Especially, as depicted in Figure 4.1, at a given operating point i_k, the neural network sensitivity of target output nodes O_T which is a subset of O is defined as

$$\delta_k = S|_{I=i_k, O=O_T} = \left.\frac{\partial O}{\partial I}\right|_{I=i_k, O=O_T}. \tag{4.2}$$

Besides, the relative (logarithmic) sensitivity is sometimes useful depending on the specific problems and defined as

$$\begin{aligned} \hat{S} &= \frac{\partial \log f(I)}{\partial \log I} \\ &= \frac{\partial \log O}{\partial \log I} \\ &= \frac{\partial O / \partial I}{O/I} \\ &= \left(\frac{O}{I}\right) S. \end{aligned} \tag{4.3}$$

Also, at a given operating point i_k, the relative sensitivity of target output nodes O_T is defined as

$$\hat{\delta}_k = \hat{S}|_{I=i_k, O=O_T} = \left(\frac{O_T}{i_k}\right) \delta_k. \tag{4.4}$$

Hence, we will focus on δ_k, the absolute sensitivity at input node k with respect to O_T, as of now in this chapter due to the representing convenience, because the relative sensitivity is obtained from $\hat{\delta}_k$ directly. Again, note that the neuron derivatives must be solved in a backward order from output to input similar to the standard back-propagation algorithm. This sensitivity analysis assumes that the neural network is fully trained and its weights are clamped.

4.1.1 Output node local gradient.

Local gradient of output node i is obtained by taking the gradient of squashing function $\varphi(net_i)$ as illustrated in Figure 4.2.

$$\delta_i = \frac{\partial o_i}{\partial net_i} = \varphi'(net_i). \tag{4.5}$$

where

$\varphi(net_i) = 1/(1 + \exp(-net_i))$, and

$\varphi'(net_i) = \varphi(net_i)(1 - \varphi(net_i))$ for the generic sigmoid squashing function.

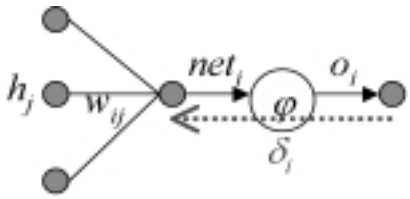

Figure 4.2: Local gradient at the output node i.

4.1.2 Hidden node local gradient.

Local gradient of hidden node j is obtained in (4.6) by chain rule collecting the gradient of its squashing function, $\varphi(net_j)$, and weighted sum of the output node local gradients described in (4.5). Figure 4.3 illustrates the procedure.

$$\delta_j = \varphi'(net_j) \sum_{i \in O} \delta_i w_{ji}. \tag{4.6}$$

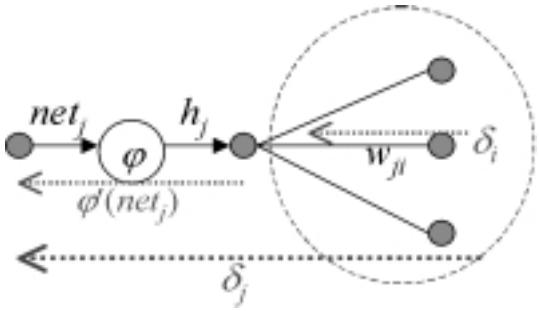

Figure 4.3: Local gradient at the hidden node j.

4.1.3 Input node local gradient.

Local gradient of input node k is obtained in (4.7) by collecting the weighted sum of the local gradient at hidden nodes in (4.6). Figure 4.4 illustrates the procedure.

$$\delta_k = \sum_{j \in H} \delta_j w_{kj}. \tag{4.7}$$

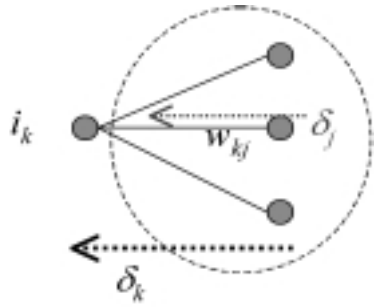

Figure 4.4: Local gradient at the input node k.

Therefore, the local gradient at the input node k with respect to output O_T is obtained as

$$\delta_k = \sum_{j \in H} \left(\varphi'(net_j) \sum_{i \in O} \varphi'(net_i) w_{ji} \right) w_{kj}. \tag{4.8}$$

(4.8) is for single hidden layer. However, the sensitivity of the neural network with multiple hidden layers can also be obtained by attaching the local gradients among hidden nodes by chain rule.

4.2 Sensitivity analysis for adaptive sonar neural network

Adaptive sonar neural network trained using SE maps of sonar data 1 is used for the sensitivity analysis. The objective of this analysis is to see how the output surveillance area, O_T, changes sensitively as each of the control parameters changes, provided that the environmental situation is fixed.

Figure 4.5 shows the absolute sensitivities of 5 control input parameters and their output changes when each input was marginally increased by Δi_k of the full scale respectively as the rest 23 environmental parameters are clamped. The original SE map at current operating point is on top of the left hand side column labeled O. O_1, O_2, O_3, O_4, and O_5 represent the SE map generated by the marginal change of the $1^{st}, 2^{nd}, 3^{rd}, 4^{th}$, and 5^{th} input parameter respectively as,

$$\begin{aligned} O_1 &= f(i_1 + \Delta i_1, i_2, i_3, i_4, i_5), \\ O_2 &= f(i_1, i_2 + \Delta i_2, i_3, i_4, i_5), \\ O_3 &= f(i_1, i_2, i_3 + \Delta i_3, i_4, i_5), \\ O_4 &= f(i_1, i_2, i_3, i_4 + \Delta i_4, i_5), \\ O_5 &= f(i_1, i_2, i_3, i_4, i_5 + \Delta i_5). \end{aligned} \tag{4.9}$$

It is more obvious that the 5^{th} control parameter is the most sensitive and the 3^{rd} the least among the others if you see the absolute error maps on the right side column of Figure 4.6, which is consistent with the sensitivity chart on top of them. This is very important information for the operator for decision making at a given situation whether he/she may take the least sensitive way of control, or vice versa.

Figure 4.6 depicts the sensitivities of 5 control parameters in a different environmental situation as well as a different operating point. In this example, the 3^{rd} control parameter is the most sensitive among the others.

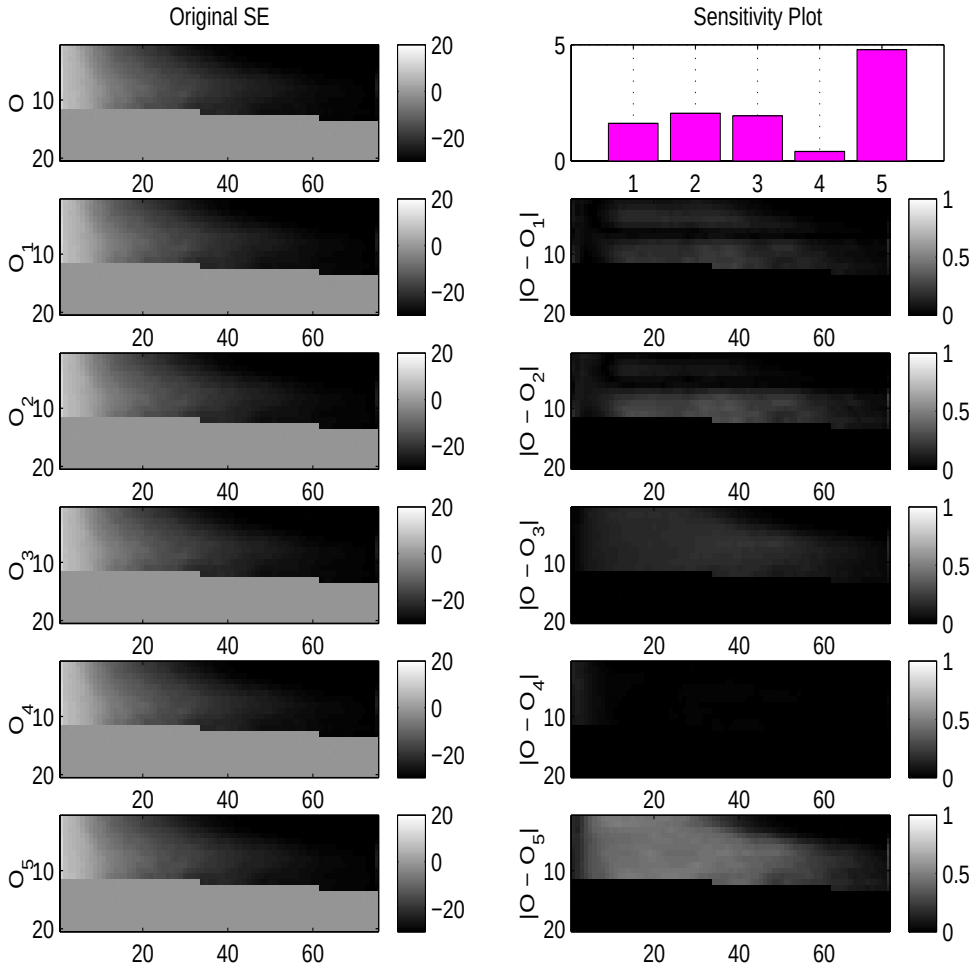

Figure 4.5: Sensitivity analysis for adaptive sonar neural network at a given environmental situation 1 (the absolute sensitivities of 5 control input parameters and their output changes when each input was marginally increased by 2% of the full scale respectively as the rest 23 environmental parameters are fixed).

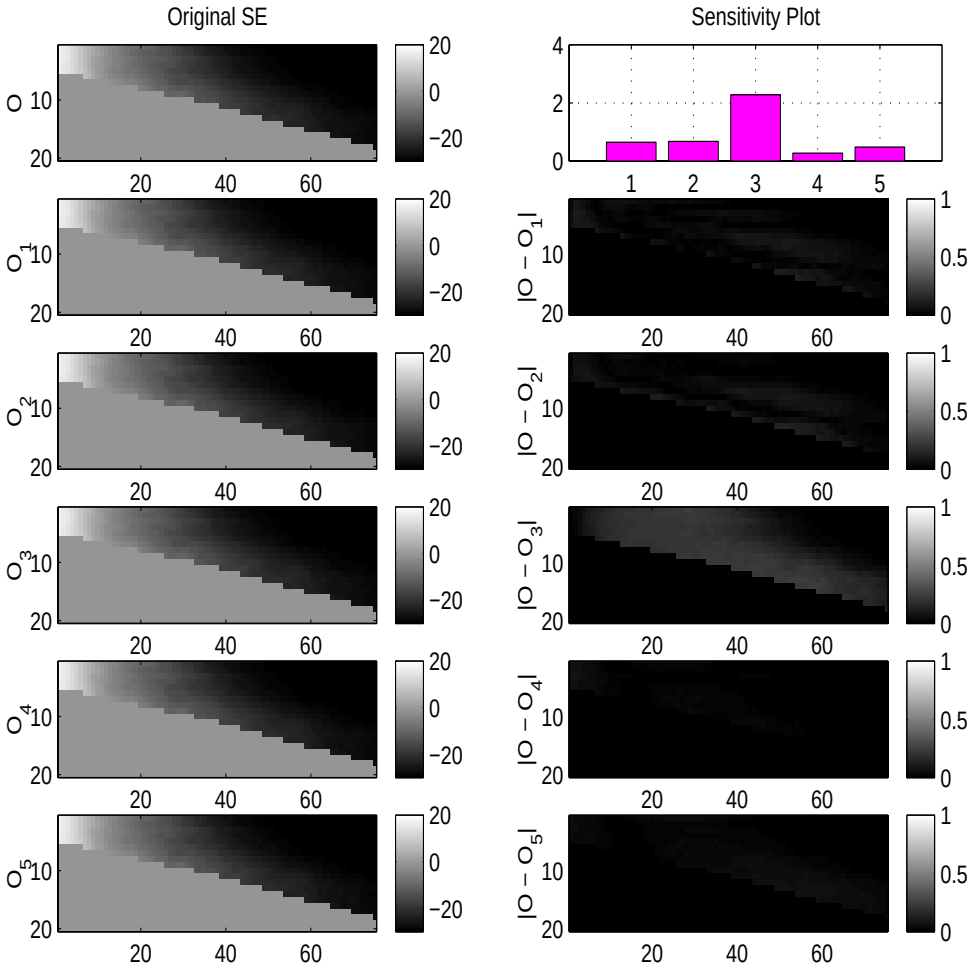

Figure 4.6: Sensitivity analysis for adaptive sonar neural network at a given environmental situation 2 (the absolute sensitivities of 5 control input parameters and their output changes when each input was marginally increased by 2% of the full scale respectively as the rest 23 environmental parameters are fixed).

Chapter 5

TEAM OPTIMIZATION OF COOPERATING SYSTEMS

When a plurality of cooperating solutions are aggregated into a single performance criterion, the set of the best component solutions is not necessarily the best set of component solutions [6], *i.e.* the best team does not necessarily consist of the best players. The composite effort of the system team, rather, is significantly more important than a single player's individual performance. We consider the case wherein each player's performance is tuned to result in maximal team performance for the specific case of maximal area coverage (MAC). The approach is first illustrated through solution of MAC by a fixed number of deformable shapes. An application to sonar is then presented. Here, sonar control parameters determine a range-depth area of coverage. The coverage is also affected by known but uncontrollable environmental parameters. The problem is to determine K sets of sonar ping parameters that result in MAC. The forward problem of determining coverage given control and environmental parameters is computationally intensive. To facilitate real time cooperative optimization among a number of such systems, the sonar input-output is captured in a feed-forward layered perceptron neural network

5.1 Preliminaries

A generic model for the team optimization of cooperating systems (TOCS) is illustrated in Figure 5.1. A total of K identical systems are replicated. The k^{th} system has control input parameters listed in the vector $\vec{c}_k$ and corresponding output response $\vec{a}_k$. The outputs are aggregated (e.g. combined). The aggregation is interpreted by a

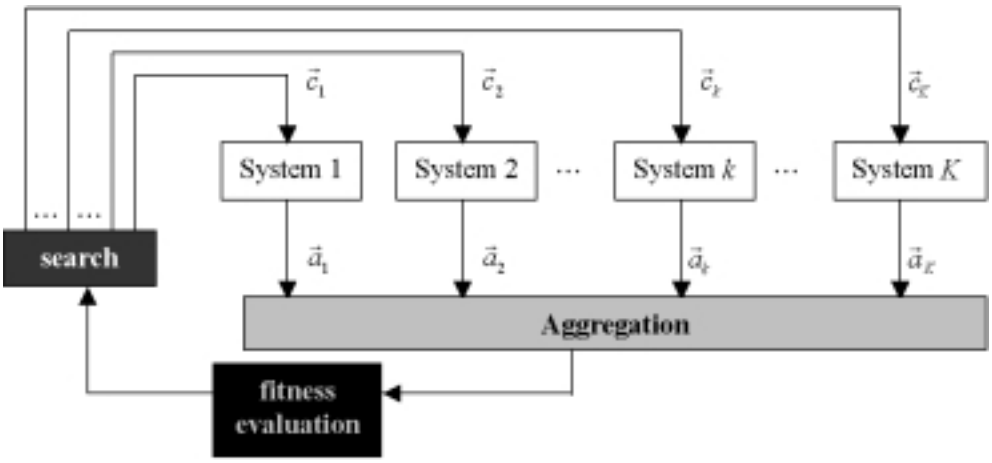

Figure 5.1: Team optimization of cooperating systems (TOCS) for maximal area coverage (MAC). The K systems, in response to stimuli generate respective area coverages of $\{\vec{a}_k\}$. These coverages are combined and a fitness function is evaluated. The fitness provides input to the control which, in turn, generates changes in the control vectors, $\{\vec{c}_k\}$.

fitness evaluation. The fitness function is used to change $\{\vec{c}_k \mid 1 \leq k \leq K\}$ in a manner that increases and ultimately maximizes the fitness measure. We use evolutionary computing for optimizing although any one of numerous optimization algorithms can also be used [34].

Although numerous combinatorial optimization problems, such as the packing problem and set-covering problem [14] can be couched in the TOCS architecture of Figure 5.1, we investigate its application only to MAC. Such problems appear in many areas. Consider, for example, the placement of cellular antennas each having area coverage controlled by tunable parameters. When locations are fixed, finding the antenna parameters to find MAC is a TOCS problem. Alternately, for antenna deployment, antenna locations can be included in the set of adjustable parameters in the TOCS. In this paper, a related problem of MAC from a sequence of sonar pings is considered.

The architecture in Figure 5.1 has the property of distributed modularity im-

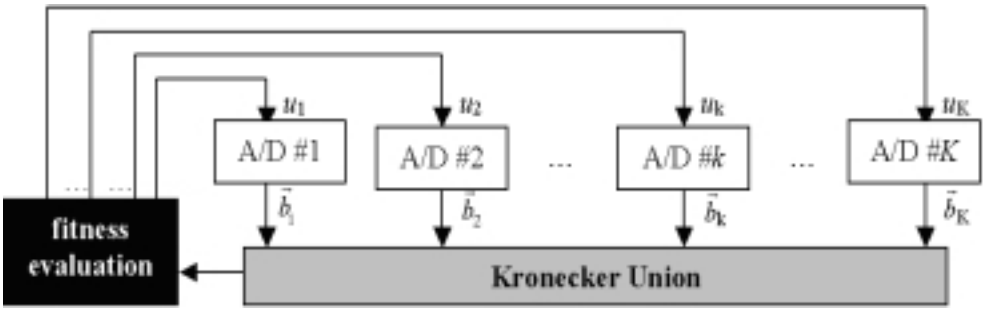

Figure 5.2: Illustration of the set covering problem.

portant when the component systems require computational intensity. The procedure also has the advantage of straightforward implementation in object-oriented languages.

5.2 Set covering problem

Set covering problems [1][40][14] are one of the typical combinatorial optimization problems. Here, we illustrate application to set covering problem as shown in Figure 5.2. K integers, $\{\vec{n}_k \mid 1 \leq k \leq K\}$, are expressed as their binary equivalents in the vectors $\{\vec{b}_k \mid 1 \leq k \leq K\}$. The Kronecker union of all the binary vectors followed by a count of the ones in the union constitutes the step of aggregation. The result of the aggregation is a scalar, a. To illustrate with toy dimension, let K=4 and $\{\vec{n}_k \mid 1 \leq k \leq K\} = \{5, 9, 11, 3\}$, then

$$\left[\vec{b}_1 \mid \vec{b}_2 \mid \vec{b}_3 \mid \vec{b}_4\right] = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{5.1}$$

The objective of the problem is to maximize a as a function of $\vec{u}$ while simultaneously

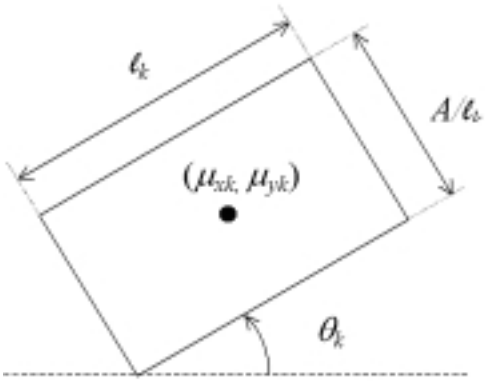

Figure 5.3: Illustration of a translatable, rotatable rectangle with adaptive aspect ratio.

minimizing the input cost as

$$\min c(\vec{u}) = \sum_{k=1}^{K} u_k n_k \tag{5.2}$$

where the solution vector, $\vec{u}$, has 1 if column k is in the solution and 0 otherwise $(k = 1, 2, \ldots, K)$ such that $u_k \in \{0, 1\}$, $k = 1, 2, \ldots, K$. The Kronecker union is equal to the logical union of elements in each row of this matrix. In this example, simple observation leads to the solution with the cost of 8 covering 3 rows $(a = 3)$.

Besides, appropriate fitness functions can be formed to satisfy $\min c(\vec{u})$ such that

$$\sum_{j=1}^{K} b_{ij} u_j \geq 1, \text{for } i = 1, 2, \ldots, K, \tag{5.3}$$

which ensures each row is covered by at lest one column. Generally, $F(\vec{u}) = a(\vec{u}) + \lambda/(c(\vec{u}) + 1)$ allows optimization along the efficient frontier as parameterized by λ.

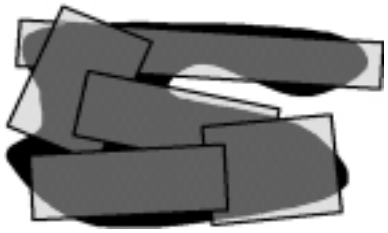

Figure 5.4: Coverage of an irregular shape with $K = 5$ rectangles of equal area.

5.3 Area coverage with deformable shapes

The MAC by TOCS procedure is applicable to area coverage with deformable shapes. To illustrate, consider the following instructive problem. K rectangular shapes of the type shown in Figure 5.3 each have fixed area, A, but can vary in accordance to aspect ratio r_k, rotation angle θ_k, and central of mass coordinates $\vec{\mu}_k = [\mu_{xk}, \mu_{yk}]^T$.

As is illustrated in Figure 5.4, the problem for a given shape is to situate K rectangles in such as a manner as to maximally cover the shape. This problem is straightforwardly implemented using the architecture in Figure 5.1. The k^{th} control vector is $\vec{c}_k = [\theta_k, r_k, \mu_{xk}, \mu_{yk}]^T$.

For example, if we assume that the area of every box is fixed to $A = l_k \cdot w_k$ and the aspect ratio is defined as $r_k = w_k/l_k$, then w_k and l_k are easily acquired as follows.

$$\begin{aligned} l_k &= \tfrac{A}{w_k} = \tfrac{A}{r_k \cdot l_k}, \quad l_k^2 = \tfrac{A}{r_k} \\ w_k &= \tfrac{A}{l_k} = \tfrac{r_k \cdot A}{w_k}, \quad w_k^2 = A \cdot r_k \end{aligned} \tag{5.4}$$

Thus, the box in Figure 5.3 can be described by 2 rectangle functions as

$$\begin{aligned} f_{xy}(\theta_k, r_k, \mu_{xk}, \mu_{yk}) = \; & \prod\left(\frac{(x-\mu_{xk})\cos\theta_k + (y-\mu_{yk})\sin\theta_k}{l_k}\right) \\ & \cdot \prod\left(\frac{(y-\mu_{yk})\cos\theta_k + (x-\mu_{xk})\sin\theta_k}{w_k}\right) \end{aligned} \tag{5.5}$$

where $l_k = \sqrt{A/r_k}, w_k = \sqrt{A \cdot r_k}$, and $\prod(\tau) = \begin{cases} 1, & \text{if } |\tau| \leq 12; \\ 0, & \text{otherwise.} \end{cases}$. Therefore, the aggregation of N different boxes is

$$g_{xy}(\theta_k, r_k, \mu_{xk}, \mu_{yk}) = \bigcup_{k}^{K} f_{xy}(\theta_k, r_k, \mu_{xk}, \mu_{yk}) \tag{5.6}$$

Now, the optimization problem is to search the best combination of K different box parameter sets so that the process maximizes the coverage determined by the intersection between the aggregation and fixed circular target area, and its evaluation function is defined as

$$Evaluation = \max(g_{xy}(\theta_k, r_k, \mu_{xk}, \mu_{yk}) \cdot t_{xy}(x_0, y_0, R)) \tag{5.7}$$

where $t_{xy}(x_0, y_0, R) = \prod\left(\frac{(x-x_0)^2+(y-y_0)^2}{2R^2}\right)$ representing the circular target area to be maximally covered, (x_0, y_0) is the center of gravity, and R is the radius of circular target area respectively. Therefore, the system outputs are representations of the areas covered by the K deformable rectangles. Aggregation is the union of these areas. The fitness value is the total area corresponding to the intersection of the aggregation with the target shape.

In order to solve this problem with large state space to be searched, genetic algorithms are used to maximize the evaluation function defined in (5.7). They belong to the class of probabilistic algorithms, yet they are very different from traditional random algorithms as they combine elements of directed and stochastic search. The operation of genetic algorithms commonly begins with a population of potential solutions (chromosomes) of the problem, and undergoes alterations by means of crossover and mutation, to obtain new generations of solutions which are better than the previous ones at least.

Examples of the evolution for MAC using TOCS is shown in Figures 5.5 and 5.6 for the cases of two and four rectangles, respectively, covering a circle. In both

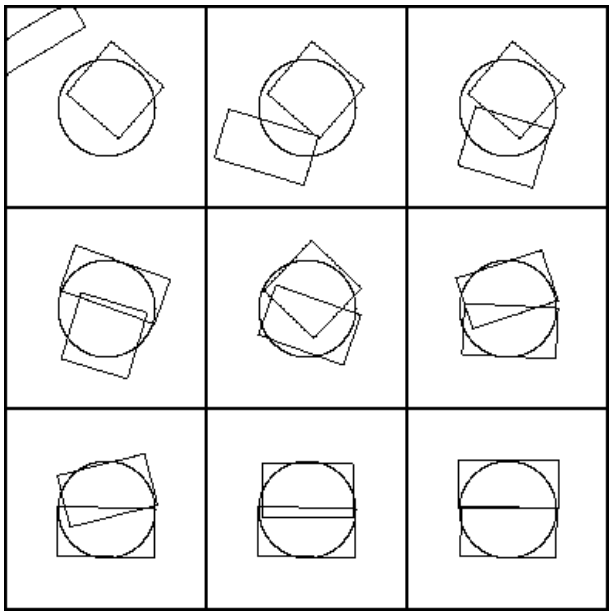

Figure 5.5: Snapshots of the process of coverage of a circle with $K = 2$ equal area rectangles. The entire circle cannot be covered. Each small block is representative of the coverage of circular target by 2 moving deformable boxes at the initial generation, 2nd generation, 15th generation, 20th generation, 35th generation, 50th generation, 587th generation, 630th generation, and the final generation, respectively, starting from upper left to lower right block.

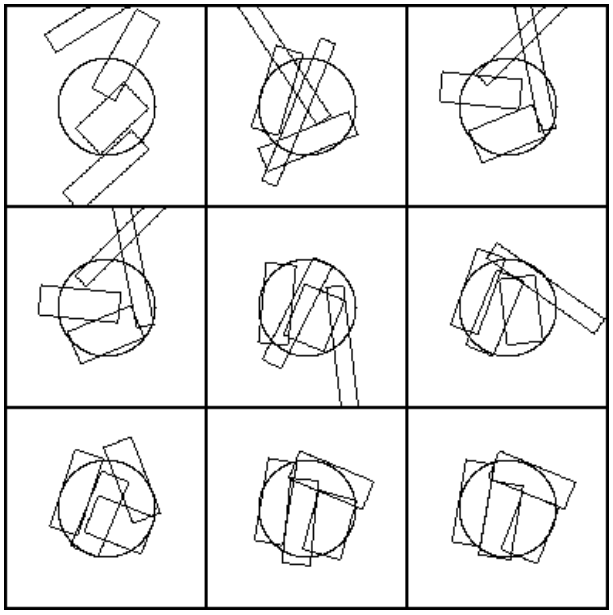

Figure 5.6: Snapshots of MAC using TOCS for 4 boxes covering a circle. Each small figure is representative of the coverage of circular target by 4 moving deformable boxes at the initial generation, 11th generation, 19th generation, 24th generation, 58th generation, 109th generation, 243rd generation, 5628th generation, and the final generation, respectively, starting from upper left to lower right block.

cases, due to the circle's symmetry, the number of solutions in the continuous version of the problem is infinite. A generic genetic algorithm was used in both cases for optimization.

5.4 Maximizing sonar area coverage

A more interesting problem of MAC using TOCS is in area coverage maximization by a plurality of sonar pings. Consider a sonar target area defined as a planar region under water of depth and range. The degree to which a single sonar ping covers a desired target area is a function of both environment and sonar control parameters. Given these parameters, computationally intensive emulations evaluate acoustic beam trajectories to evaluate acoustic signal strength at a plurality of depths and ranges.

Relevant environmental parameters include wind speed and bottom type, bathymetry, and sound speed profile. The positions of the submerged sonar transmitter and receiver also affect the ensonification. These two depths, in addition to the sonar ping parameters, constitute the control parameters of the problem. The sonar parameters can be controlled. The environmental parameters can't.

The sonar problem under consideration is this. For a given environment, we desire, with K pings of the sonar, to observe as large an area in range and depth as possible. This is equivalent to design of the K ping parameters, $\{\vec{c}_k \mid 1 \leq k \leq K\}$, in Figure 5.1. The systems in Figure 5.1 are the emulators that, for given environmental conditions, compute coverage of the sonar. In range and depth, the output of each system in Figure 5.1 is the signal excess (SE) corresponding to the imposed control parameters. The SE is akin to a signal to noise ratio and measures the ability to detect a target. When the SE exceeds a prescribed threshold, coverage is assumed. Otherwise, it is not. The individual outputs can be thus characterized as binary maps - one for where coverage is made and zero otherwise.

As in the case of the previous example, aggregation is the union of the binary maps emerging from each of the systems. The fitness of the control parameters is equal to the total area covered by the K sonar pings.

The forward problem for each of the K system modules in Figure 5.1 is performed using Applied Physics Laboratory Acoustic Simulation Software (APLASS) [8]. The APLASS software is computationally intensive and several minutes are required to analyze the forward problem: the signal excess in range and depth as a function of the input environmental and control parameters. In order to speed up the optimization, APLASS data was used to train a layered perceptron[1]. The result is that the perceptron, after proper training, emulates the same results as APLASS - but much more quickly. K identical neural networks trained to emulate APLASS are then used

[1]Details of the training of the neural network using APLASS data is given by Reed [34], Jensen et. al.[8] and Jung et. al.[9]

in the architecture in Figure 5.1.

The resulting optimal control parameters in conjunction with environmental parameters will maximize the combined coverage and individual coverage maps and corresponding cumulative maps as is illustrated in Figure 5.7. The outputs from each bank can be characterized as binary maps - one for where coverage is made and zero otherwise. As in the case of the previous example, aggregation can be the union of the binary maps emerging from each of the systems. The fitness of the control parameters is equal to the total area covered by the K pings. The neural networks denoted as 1 to K in Figure 5.6 are identical but placed in parallel to distinguish different outcomes. The fitness function is determined by appropriate aggregation procedure from the K output signal excess (SE) maps $(\vec{O}_1, \vec{O}_2, ..., \vec{O}_K)$, which are acquired from a set of K input vectors whose elements include a sonar control parameter and fixed environmental parameters. The aggregation procedure translates K output SE maps into a value that corresponds to the sonar ping coverage to be compared with desired coverage performance.

The MAC procedure consists of three steps of operations. The maximum SE map, $\vec{O}_{max}$, is calculated by taking maximum value in each element of K output SE maps respectively.

$$O_{max,j} = \max_{k=1}^{K} O_{k,j} \tag{5.8}$$

where $O_{k,j}$ is the j^{th} element of the k^{th} output SE map, $\vec{O}_k$, and $O_{max,j}$ is the j^{th} element of maximum SE map $\vec{O}_{max}$. This maximum SE map is fed into the nonlinear squashing function. Hence, in lieu of a strict binary representation, every element in the maximum SE map lies between 0 and 1 based on the specific threshold value implying whether the SE values are large enough to be considered covered or not[2].

[2]Such representation also allows the use of gradient based search techniques.

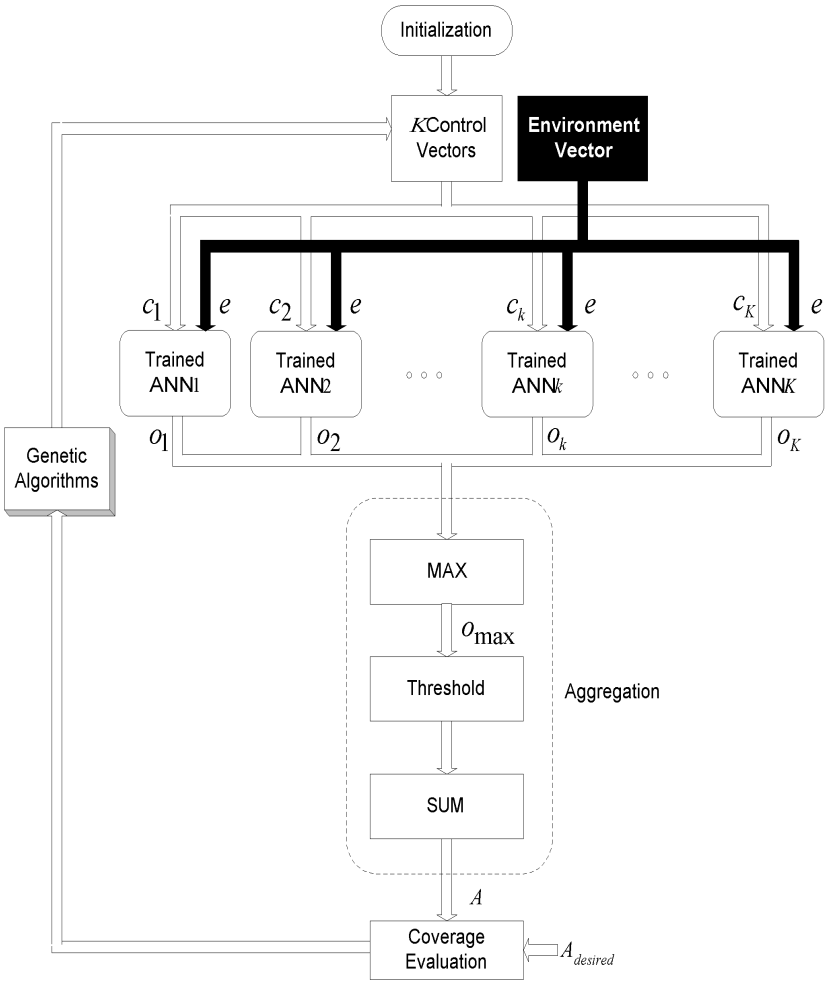

Figure 5.7: MAC by TOCS as applied to sonar. The vector of environmental parameters, $\vec{e}$, is fixed. For the given environmental parameters, the combination of control vectors, $\{\vec{c}_k \mid 1 \leq k \leq K\}$, giving a combination maximal area of ensonification are desired. The control vector contains the parameters to be varied. The overall fitness value is equal to the area covered by the ensonification. A generic genetic algorithm is used to perform the search over the K vectors.

The j^{th} element of the maximum SE map, $\vec{O'}_{max}$, is denoted as

$$O'_{max,j} = \frac{1}{1 + exp^{-\alpha(O_{max,j} - \theta)}} \tag{5.9}$$

where α is a sensitivity parameter of sigmoid slope, and θ is a prespecified soft threshold value[3]. All elements in the composite maps are summed up to give a global coverage

$$A = \sum_{j=1}^{N} O'_{max,j} \tag{5.10}$$

Accordingly, the fitness is calculated by normalizing the resulting aggregation with the desired aggregation, $A_{desired}$, which is N.

$$Fitness = \frac{A}{A_{desired}} \tag{5.11}$$

For example, assuming 4 sonar pings problem (K=4), a single sonar control parameter, sonar depth, is implemented by bit stream with required precision of places after the decimal point. Thus, the required number of bits for each depth is 17. (See Figure 5.8).

Several genetic algorithm emulations using different probabilities are performed and their fitness functions converged in all cases to the same values. The algorithm, for this problem, was remarkably insensitive to initialization and algorithm parameters. Convergence for a number of cases is shown in Figure 5.9 as a function of generation.

The resulting optimal sonar control parameters in conjunction with environmental parameter maximize the combined global sonar ping coverage. Physical limitations imposed by the fixed environment prohibits 100% of target surveillance area no matter

[3]Alternately, if strictly binary maps are desired, a hard limiter can be used.

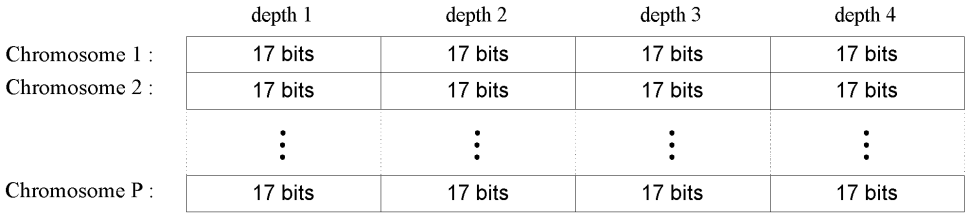

Figure 5.8: Population of genetic algorithm for sonar ping coverage optimization.

how many pings are used. Figure 5.10 illustrates coverage maps of best 4 sonar pings and their contributions (cumulative coverage) as each sonar ping coverage map is added.

Figure 5.11 shows an experiment with SE map of sonar data 1. Best 2 sonar control sets are searched through the genetic algorithm to maximize the combined sonar coverage. The first row contains a set of 5 control parameters which is one of the best 2 sets, SE map reproduced by that control set coupled with current environment, and its binary coverage map. The next row has the other control set, SE map, and its coverage map, respectively. The last row includes the convergence map, the combined SE map taken by pixel-wise maximum from 2 SE maps above, and the cumulative coverage map, respectively.

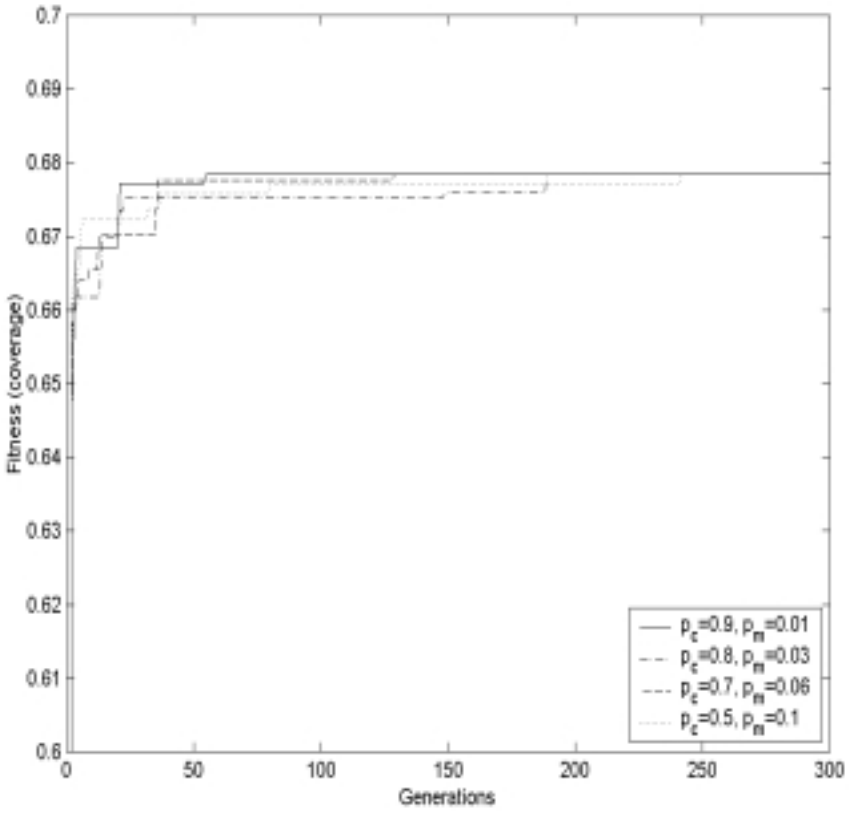

Figure 5.9: Fitness function convergence using different probabilities of crossover and mutation.

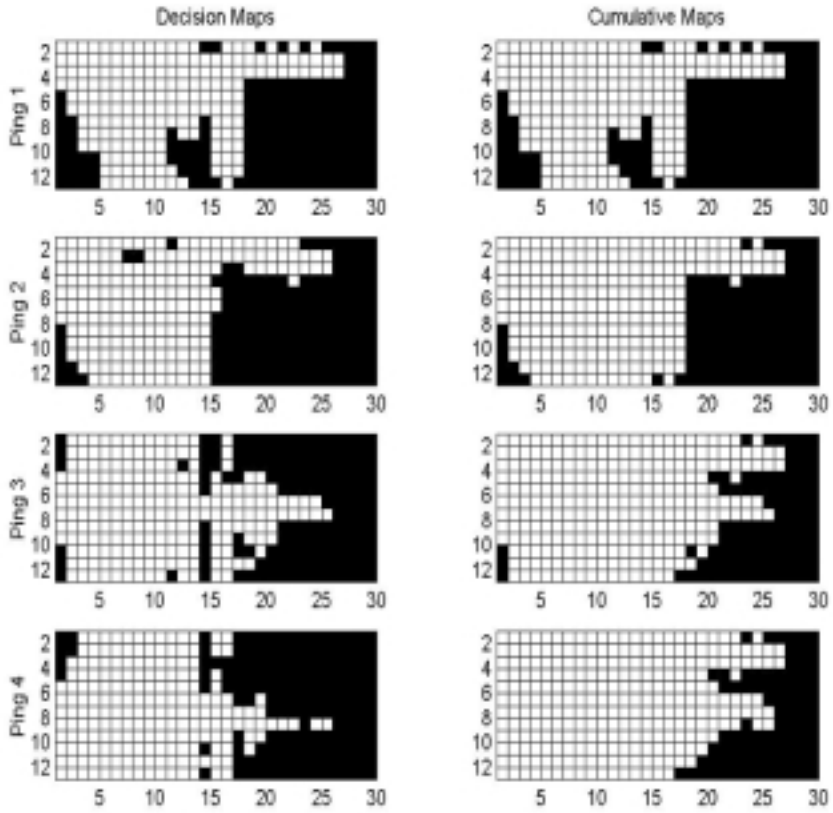

Figure 5.10: Modular Maximal Area Coverage Using a Neural Network Bank for Multiple Sonar Ping Problem (Best 4 different sonar pings contribute the maximal coverage after the convergence of GA fitness evaluations)

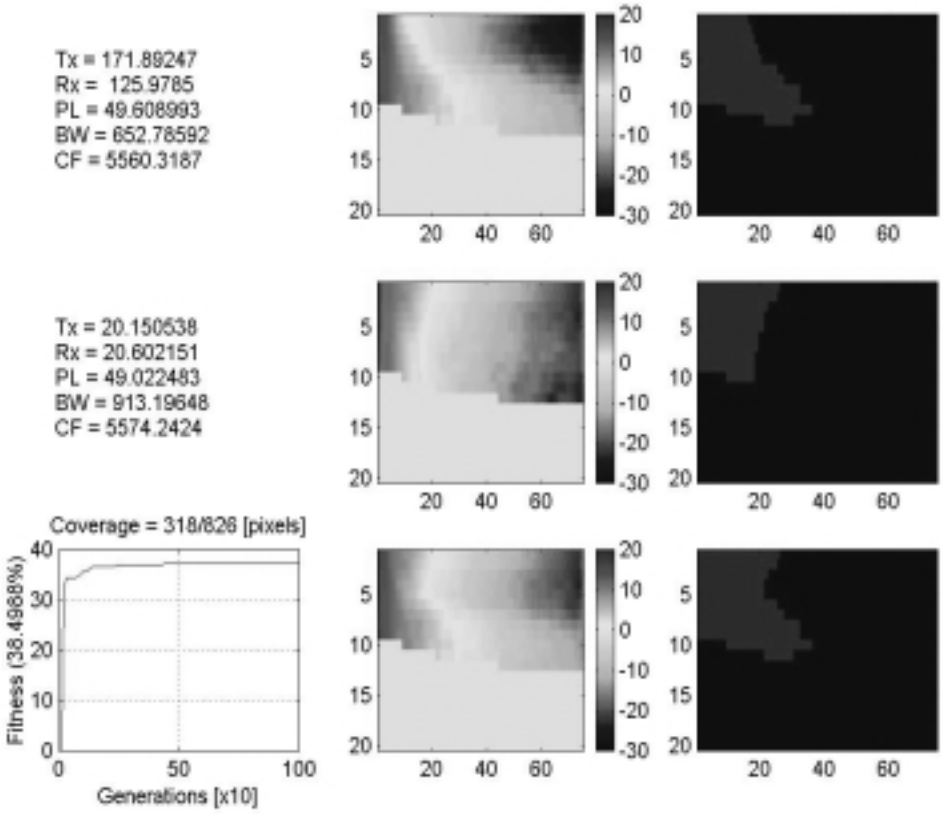

Figure 5.11: an experiment with SE map of sonar data 1. Best 2 sonar control sets are searched through the genetic algorithm to maximize the combined sonar coverage.

Chapter 6

CONCLUSIONS

6.1 Contributions

This dissertation has made several contributions to the state of the art including:

- A novel neural network learning algorithm for data sets with varying output dimension is proposed in this paper. The possible memorization problem caused by irregular weight correction is avoided by employing step size modification.

- A new neural network inversion algorithm was proposed whereby several neural networks are inverted in parallel. Advantages include the ability to segment the problem into multiple sub-problems which each can be independently modified as changes to the system occur over time. The concept is similar to the mixture of experts problem applied to neural network inversion.

- The sensitivity of neural network is investigated in this work. Sensitivity analysis is very useful. Once neural network is trained, especially, it is very important to determine which of the control parameters are critical to the decision making at a certain operating point such that environmental situation or/and control criteria is given. This can be done through input parameter sensitivity analysis.

- There exist numerous generalizations of the fundamental architecture of maximal area coverage problem that allow application to a larger scope of problems.

    1. The systems need not be replications of each other but can, for example, specialize in different aspects of appeasing the fitness function.

2. The search can be constrained[8]. In Figure 5.1, for example, a constraint imposing module can be inserted between the search box and the inputs to the systems. A simple example of constraint imposition is requirement that each element of each lie within specified operating limits.

6.2 Ideas for future work

Several areas exist for future work including:

- More work could be done for more accurate training of sonar data, especially, if the decision logic associated with the probability of detection [%] is important instead of signal excess [dB]. Multi resolution neural networks could help extract discrete detection maps.

- Data pruning using nearest neighbor analysis before training or query-based learning using sensitivity analysis during training could improve the training time or/and accuracy.

- Extensive research for the use of evolutionary algorithms to improve the inversion speed and precision. Particle swarm optimization or genetic algorithms could be considered for more flexibility on imposing feasibility constraints.

BIBLIOGRAPHY

[1] M. Aourid and B. Kaminska. Neural networks for the set covering problem: An application to the test vector compaction. IEEE World Congress on Computational Intelligence, 7:4645–4649, 1994.

[2] Jeng-Neng Hwang Avni H. Rambhia, Robb Glenny. Critical input data channels selection for progressive work exercise test by neural network sensitivity anaysis. IEEE International Conference on Acoustics, Speech, and Signal Processing, 2:1097 –1100, 1999.

[3] G.A. Carpenter and S. Grosberg. Art2: Self-organization of stable category recognition codes for analog input patterns. Applied Optics, 26:4919–4930, 1987.

[4] G.A. Carpenter and S. Grosberg. A massively parallel architecture for a self-organizing neural network recognition machine. Computer Vision, Graphics, and Image Processing, 37:54–115, 1987.

[5] Mandira Chakraborty and Uday K. Chakraborty. Applying genetic algorithm and simulated annealing to a combinatorial optimization problem. International Conference on Information, Communications and Signal Processing, pages 929–933, September 1997.

[6] T.M. Cover. The best two independent measurements are not the two best. IEEE Transactions on Systems, Man and Cybernetics, 4:116–117, January 1974.

[7] Russell Eberhart and James Kennedy. A new optimizer using particle swarm

theory. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pages 39–43, 1995.

[8] C.A. Jensen et. al. Inversion of feedforward neural networks: Algorithms and applications. Proceedings of the 1999 IEEE International Conference on Robotics and Automation, 87:1536–1549, September 1999.

[9] Jae-Byung Jung et. al. Neural network training for varying output node dimension. International Joint Conference on Neural Networks, 3:1733–1738, July 2001.

[10] Jae-Byung Jung et. al. Team optimization of cooperating systems: Application to maximal area coverage. International Joint Conference on Neural Networks, 3:2212–2217, July 2001.

[11] Jenq-Neng Hwang et. al. Query-based learning applied to partially trained multilayer perceptrons. IEEE Transactions on Neural Networks, 2(1):131–136, January 1991.

[12] Laurene Fausett. Fundamentals of Neural Networks. Prentice Hall, 1994.

[13] D. B. Fogel. Evolutionary computation: toward a new philosophy of machine intelligence. IEEE Press, 1995.

[14] Mitsuo Gen and Reunwei Cheng. Genetic Algorithms and Engineering Optimization. Wiley-Interscience, 2000.

[15] Jr. Gerald W. Davis. Sensitivity analysis in neural net solutions. IEEE Transactions on System, Man, and Cybernetics, 19(5):1078–1082, September 1989.

[16] D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1998.

[17] Jurgen Ihlow Haoxun Chen and Carsten Lehmann. A genetic algorithm for flexible job-shop scheduling. Proceedings of the IEEE, pages 1120–1125, May 1999.

[18] Bart L.M. Happel and Jacob M.J. Murre. Design and evolution of modular neural network architecture. Neural Networks, 7(6):985–1004, 1994.

[19] Sherif Hashem. Optimal linear combinations of neural networks. Neural Networks, 10(4):599–614, 1997.

[20] Simon Haykin. Neural Networks. The IEEE Press, 1994.

[21] R.A. Jacob and M.I. Jordan. Learning piecewise control strategies in a modular neural network architecture. IEEE Transactions on System, Man, and Cybernetics, 23(2):337–345, 1993.

[22] R.A. Jacob and M.I. Jordan. Hierarchical mixtures of experts and em algorithm. Neural Computaion, 6:181–214, 1994.

[23] L.C. Jiao and Z. Bao. Neural networks via nonlinear sensitivity approach. IEEE International Sympoisum on Circuits and Systems, 5:2542 –2545, 1991.

[24] Nandakishore Kambhatla. Dimension reduction by local principal component analysis. Neural Computation, 9:1493–1516, 1997.

[25] Juha Karhunen and Jyrki Joutsensalo. Generalizations of principal component analysis, optimization problems, and neural networks. Neural Networks, 8(4):549–562, 1995.

[26] James Kennedy and Russell Eberhart. Particle swarm optimization. IEEE International Conference on Neural Networks, 4:1942–1948, 1995.

[27] T. Kohonen. Self-Organization and Associative Memory. Springer Series in Information Science 8, Heidelberg, 1984.

[28] A. Linden and J. Kindermann. Inversion of multilayer nets. Proceedings of the International Joint Conference on Neural Networks, 2:425–430, 1989.

[29] A. Linden and J. Kindermann. Inversion of neural networks by gradient descent. Parallel Computing, pages 277–286, 1990.

[30] Zbugniw Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer, 1996.

[31] Erkki Oja. Principal component analysis by homogeneous neural networks. IEICE Trans. Inf. and Syst., E75-D(3):366–382, 1992.

[32] Ozcan and Chilukuri K. Mohan. Particle swarm optimization: Surfing the waves. Proceedings of the 1999 Congress on Evolutionary Computation, 3:1939–1944, 1999.

[33] A. Papoulis. Probability, Random Variables, and Stochastic Processes. McGraw-Hill, 1991.

[34] R.D. Reed and R.J. Marks II. Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. MIT Press, 1999.

[35] Elaine Rich and Kevin Knight. Artificial Intelligence. McGraw-Hill, 1991.

[36] John Salerno. Using the particle swarm optimization technique to train a recurrent neural model. IEEE International Conference on Tools with Artificial Intelligence, pages 45–49, 1997.

[37] M.A. El-Sharkawi Seho Oh, R.J. Marks II. Querry-based learning in a multilayered perceptron in the presence of data jitter. Proceedings of the First International Forum on Applications of Neural Networks to Power Systems, pages 72–75, 1991.

[38] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. IEEE World Congress on Computational Intelligence, pages 69–73, 1998.

[39] Georg Thimm and Emile Fiesler. High-order and multilayer perceptron initialization. IEEE Transactions on Neural Networks, 8(2):349–359, 1997.

[40] Jorng-Tzong Horng Wen-Chih Huang, Cheng-Yan Kao. A genetic algorithm approach for set covering problems. IEEE World Congress on Computational Intelligence, 2:569–574, 1994.

[41] R.J. Williams. Inverting a connectionist network mapping by backpropagation of error. 8th Annual Conference of the Cognitive Science Society, pages 859–865, 1986.

[42] Jacek M. Zurada. Artificial Neural Systems. West Info Access, 1992.

Appendix A

SONAR DATA

Applied Physics Laboratory's acoustic simulation software is computationally very intensive and even several minutes are required to analyze a forward problem. The signal excess (SE) map or transmission loss (TL) map in range and depth as a function of both environmental and control parameters is obtained through the sumulation of sonar acoustic propagation. Two different versions of sonar data sets were used in this work. One is more complex and realistic and the other is simplified version.

A.1 Sonar data 1

A surface ship controls the depth to which both transmitter and receiver are submerged, pulse length, band width, and center frequency. The environmental parameters include 19 measurable physics (e.g., total noise, target echo duration, wind speed, volume scattering strength, bottom type and sound speed profile) and 3-point bathymetry listed in Table A.1.

The surveillance area assigned to the sonar is shown in Figure A.1. For analysis, the surveillance area is divided into pixels. There are 1500 signal excess values in [dB] corresponding to 20 depth pixels by 75 range pixels covering, respectively, a surveillance range of 400m by 15km. A dark region on the bottom is ocean floar described by 3-point bathymetry where signal excess can not be defined.

Figure A.2 shows a sample transmission loss (TL) map which has exactly same dimension as signal excess map but different acoustic characteristics.

Table A.1: Environmental and control parameters used as inputs to train the neural network.

No	Input parameter	No	Input parameter
1	Transmitter depth [m]	15	Sound speed at surface
2	Receiver depth [m]	16	Sound speed at 10 [m/s]
3	Pulse Length [s]	17	Sound speed at 20 [m/s]
4	Bandwidth [Hz]	18	Sound speed at 30 [m/s]
5	Center Frequency [Hz]	19	Sound speed at 50 [m/s]
6	Total Noise [dB]	20	Sound speed at 75 [m/s]
7	Target echo duration [s]	21	Sound speed at 100 [m/s]
8	Wind speed [m/s]	22	Sound speed at 125 [m/s]
9	Volume scattering strength [dB/m]	23	Sound speed at 150 [m/s]
10	Bottom type, grain size[mm]	24	Sound speed at 200 [m/s]
11	Bathymetry 1 (depth at sonar) [m]	25	Sound speed at 250 [m/s]
12	Bathymetry 2 (range in the middle) [m]	26	Sound speed at 300 [m/s]
13	Bathymetry 3 (depth in the middle) [m]	27	Sound speed at 400 [m/s]
14	Bathymetry 4 (depth at 15 km) [m]	28	Sound speed at 500 [m/s]

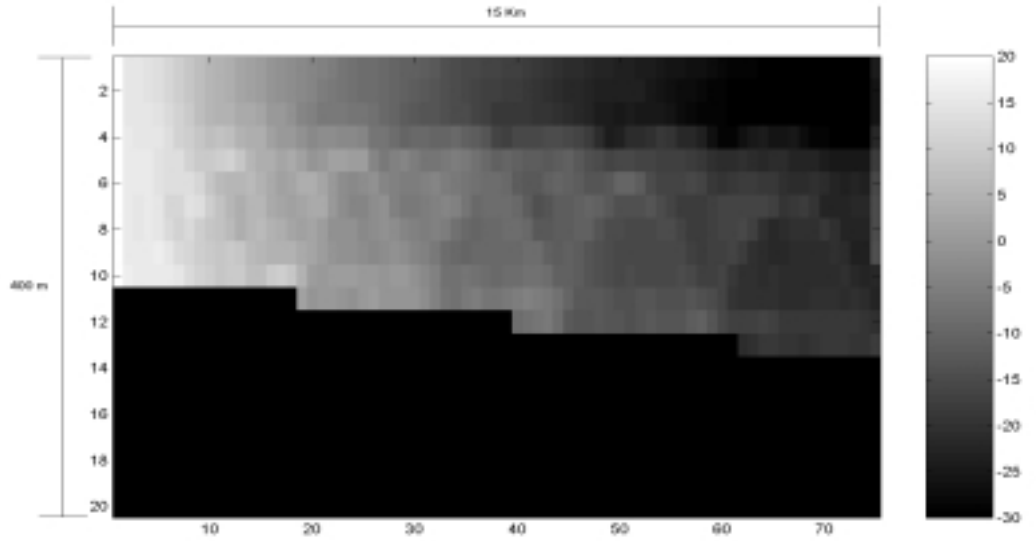

Figure A.1: A sample SE map.

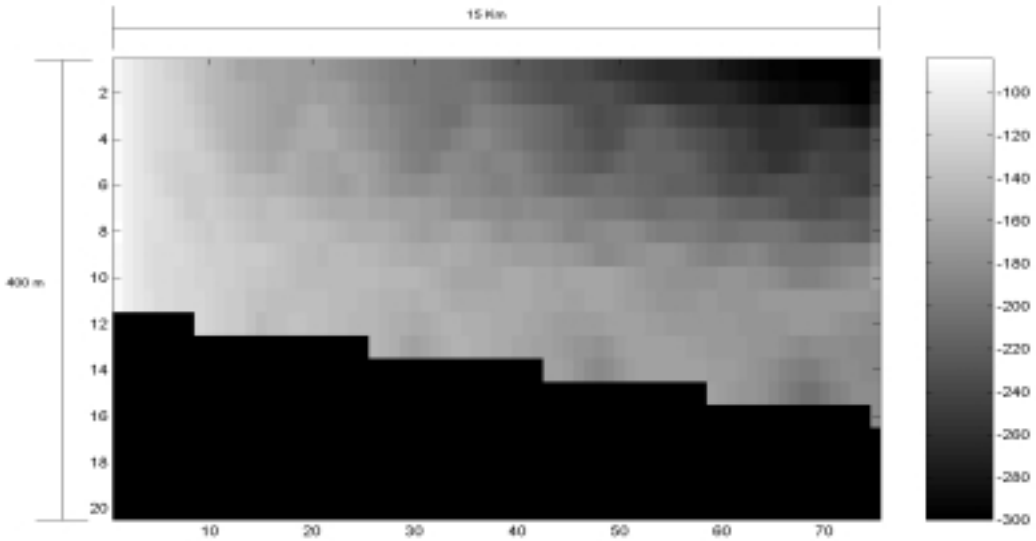

Figure A.2: A sample TL map.

Table A.2: Environmental and control parameters used as inputs to train the neural network - simplified version.

No	Input parameter
1	Transmitter/Receiver depth [m]
2	Wind speed [m/s]
3	Sound speed at surface [m/s]
4	Sound speed at bottom [m/s]
5	Bottom type, grain size[mm]

A.2 Sonar data 2

In this scenario, both the input and output are simplified. A surface ship now controls only the depth to which both transmitter and receiver are submerged. The environmental parameters include wind speed, bottom type, sound speed at surface and sound speed at bottom, which are listed in Table A.2. The surveillance area assigned to the sonar is shown in Figure A.3. For analysis, the surveillance area is divided into pixels. There are 390 signal excess values in [dB] corresponding to 13 depth pixels by 30 range pixels covering, respectively, a surveillance range of 200m by 6km. No ocean floar is included in the surveillance area.

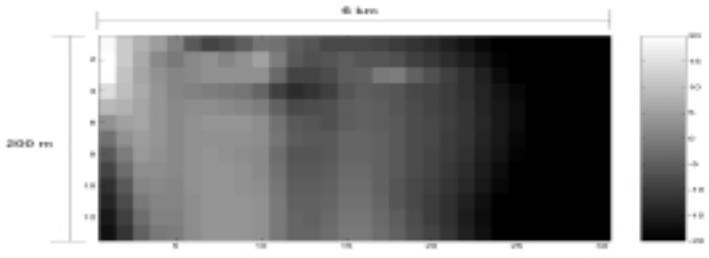

Figure A.3: A sample SE map - simplfied version.

VITA

Jae-Byung Jung was born on November 2, 1970 in Seoul, Korea. He attended Hanyang University and received the B.S. and M.S. degrees in 1993 and 1995 respectively. He has worked for LG Industrial System Company in Korea from 1995-1996 as an research engineer. In 2001 he earned a Doctor of Philosophy at the University of Washington in Electrical Engineering. His research interests include computational intelligence, neural networks, evolutionary programming, fuzzy systems, robotics and computer vision.