This is to certify that the

dissertation entitled

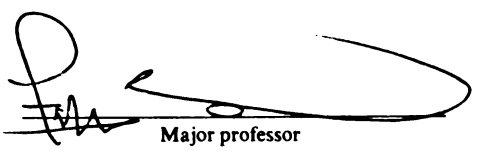
EFFICIENT EXTENDED KALMAN FILTER
LEARNING FOR FEEDFORWARD
LAYERED NEURAL NETWORKS

presented by

Saida Benromdhane

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering



Major professor

Date April 29, '96

Efficient Extended Kalman Filter Learning for Feedforward Layered Neural Networks

By

Saida Benromdhane

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1996

ABSTRACT

Efficient Extended Kalman Filter Learning for Feedforward Layered Neural Networks

By

Saida Benromdhane

The thesis focuses on the computationally efficient convergence to satisfactory local minima of the Extended Kalman Filter Algorithm (EKF) when it is used in the supervised learning of Artificial Feedforward Neural Networks. There are two stages to our research work.

In the first stage, the effect of different choices of the energy parameter or weighting factor λ on the convergence of the EKF algorithm is investigated. We limit our attention to problems related to the supervised learning of Feedforward Artificial Neural Networks. Through the simulation of two region classification problems and the analysis of the results, we demonstrate that when λ is chosen slightly smaller than 1, the algorithm experiences explosive divergence : The Least Square Error (LSE) grows indefinitely. However, for a choice of λ slightly greater than 1, the algorithm is stable but often converges to unsatisfactory local minima, from the point-of-view of performance and computation time.

The second stage of our work is where we propose several modifications of the

algorithm. These modifications are aimed at improving the efficiency of the algorithm both in terms of performance and speed of convergence. One modification in the algorithm is the development of an update mechanism for the exponential weighting factor λ which self-adjusts to the (LSE). The second modification is an augmentation of the recursion formulae of the algorithm. Both of these modifications result in a significant improvement in performance as well as marked decrease in convergence time when compared with the original EKF algorithm.

To my father, who always prayed for my success but could not live to see this day.

ACKNOWLEDGEMENTS

In The Name Of Allah
Most Gracious, Most Merciful.

All praises be to Allah *Subhanahu-wa-Taala*, the creator and sustainer of this universe for providing me the strength and the patience to complete this work.

I am greatful for the financial support from the Government of Tunisia, and the department of Electrical Engineering at Michigan State University. Without this support, this research work would not have been accomplished.

I would like to express my sincere thanks to my major advisor, Dr. F. M. A. Salam for his continued support and guidance throughout the years of my graduate studies. I also would like to thank the members of my committee, Dr. Hassan Khalil, Dr. Joel Shapiro, and Dr. Majid Nayeri for their helpful criticism and suggestions.

I wish to express my greatest thanks and appreciation to my parents for their continued prayers, support and encouragement. I deeply owe my heartiest feelings of gratitude to my husband, Izzat, for his patience and support especially during the last years of my research. The warmest feelings of gratitude are owed to my sisters Souad and Sonia, and my brothers Lassaad, Sofiane and Souheil. They have always prayed for my success and supported my pursuit of a Doctoral degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Even though Artificial Neural Networks may be limited and imperfect, they are still one of the few paradigms, if not the only ones, that seek to mimic natural intelligence at an architectural level. They have performed impressively in some limited applications. The success and failure of Neural Networks can only suggest that research will ultimately give rise to artificial systems that can perform the same tasks that only the humans are currently able to perform. That is what constitutes the major drive behind the exponential growth of Neural Networks research.

An outcome of the renewed interest in Neural networks is their use in numerous applications ranging from pattern classification and completion to automatic control. The network paradigms used in these applications can be very distinct each carrying their own name. They are made of a large number of processing elements operating in parallel. The topology that connects these processing elements is what determines the distinction between two or more network paradigms.

Given that the main goal for Artificial Neural Networks is to possess human intelligence, the inspiration for research had to come from more specifically from neuro-biology. Neuro-biology offers a cellular formulation of the principles of intelligent behavior which inspire researchers to construct better intelligent systems known as "Artificial Neural Networks".

1.1 Inherent Properties of Neural Networks

This new hope finds its origins in all the inherent properties of Neural Networks, but more so in these four particularly important ones: (1)generalization (2) graceful degradation (3)adaptivity and learning and (4)parallelism [1, 2]. A focus on these properties will be sufficient to illustrate the importance of Neural Networks for related fields.

Given that Neural Networks emulate continuous activation units, they provide a continuous representation and inference system with similarity metrics such that when similar real-world data inputs are presented, outputs or inferences are also similar. A smooth generalization from stored cases to new ones is then expected.

The generalization performance of logic-based systems deteriorates considerably when the data available is inaccurate or incomplete. In contrast, the generalization performance of Neural Networks deteriorates proportionally to the degree of inaccuracy or incompleteness of the data, which is usually small. However, what really makes Neural Networks distinct from other expert systems are the two last characteristics mentioned above. The first one manifests itself in the way Neural Networks learn and adapt. Neural networks are taught by presentation of examples. They also adapt very easily to changing conditions. This issue is still an unsolved problem with other expert systems. The second characteristic is present in the architecture of Neural Networks adaptation and learning which are conducted totally in a parallel structure. There are other parallel reasoning models that have been developed and implemented, but in most cases they have to be supported by the user. By contrast, in Neural Networks, units in the same layer can be updated simultaneously. In this way, computationally expensive tasks can be performed efficiently.

1.2 Why a Different Training Approach?

For several years the back propagation algorithm (BP) which is based on the gradient descent algorithm has been widely used to train Multi-Layered Neural Networks(MNNs) to perform a desired task. The common use of the back propagation algorithm is attributed to its strengths which include a generalization ability from a modest number of training patterns, an acquiring of arbitrarily complex nonlinear mappings, all within a parallel structure of computation. However, like any other algorithm, the back propagation algorithm has limitations that are easily recognized during its use. The first noticeable one is the slow convergence time. The second limitation is an off-line encoding requirement which makes it unsuitable for certain applications, especially when dealing with temporal signals.

In order to improve the speed of convergence of the (BP) algorithm, numerous schemes have been suggested, for example [3, 4, 5]. One of these schemes [5] used different energy functions to achieve faster convergence of the (BP)algorithm. All of these enhancements are computationally inexpensive, but they often require tunable parameters, thereby adding cumbersome guesswork.

Given that the enhanced backpropagation algorithm, like the standard one, may not adequately handle temporal signals, it would not be suitable for some specific applications, for instance, the identification of temporal signals. In the face of this limitation, the problem that poses itself is how to formulate a Feedforward Artificial Neural Networks algorithm that possesses the temporal capability and is also efficient in terms of convergence time, guesswork and computation. In the last few years much interest focused on training algorithms[6, 7] that are faster than the back-propagation algorithm or are capable of solving more complex problems. A class of alternative algorithms that have been used in different applications is based on the Extended Kalman Filter algorithm. The Kalman algorithm, in contrast to gradient techniques,

computes the optimum value of the network parameters each time a new data point is presented, therefore it is well suited for temporal signal processing.

1.3 The Contribution of the Thesis

The main topic of this thesis is concerned with the Extended Kalman Filter Approach for training Feedforward Artificial Neural Networks FFANN. This is a supervised learning which is different from the backpropagation.

Motivated by the help of previous and current analysis [8], a framework supported by computer simulations is developed to improve the supervised learning of FFANN using the Kalman Filter approach. The framework encompasses modifications of the Extended Kalman Filter algorithm including the development of an update mechanism for the exponential weighting factor λ [8]. This mechanism self-adjusts according to the mean square error LSE. The weighting factor is also implemented as a function of the error. The update is aimed to speed up and enable convergence to minima that are satisfactory from the point-of-view of performance. A different computation of the gain matrix is also pursued.

A final and important added feature to the algorithm is the augmentation of its recursion formulas. A recursion formula for the slope of the sigmoid nonlinearity is developed where the update is based on the gradient descent rule. The gradient descent rule is known to decrease the MSE without adding any oscillations.

CHAPTER 2

Background

The inspiration for Neural Networks structures is mainly based on our present understanding of the nervous biological system. To completely specify a Neural Network model, its physical parts have to be defined along with the required algorithms. Similar to any other network a Neural Network is comprised of nodes we call neurons as its processing units, and edges connecting the neurons therefore defining the topology of the network. Once a network's topology is defined, this Neural Network is classified in one of several Neural Networks models. These models include the feedforward neuron-model, the feedback neuron-model and a completely interconnected neuron-model. Our primary interest in this work is in the feedforward neuron-model and the topology that defines the connection between its neurons. In the following two sections the feedforward neuron model and its topology will be discussed in detail. The resulting network will be referred to as the Feedforward Artificial Neural Network(FFANN)

In the design of a Neural Network model, defining the topology between the neurons is only the first step. Devising the learning rule for this network is the most challenging part in the design. The learning rule is the core constituent of the algorithm necessary for the operation of the neurons. It consists of changing the state of individual neurons using inputs from neighboring neurons. This rule can take

various particular forms, therefore we will only emphasize the forms that have been used in (FFANN). We should also mention the one feature that is common to many models: *most learning rules can be viewed as modifications of the stochastic gradient optimization of a certain objective function or performance criterion.* An algorithm where this feature is very well known is the error backpropagation algorithm. Our primary interest in this thesis is another model that also has this feature and is a (FFANN) where the learning is based on the Kalman optimal filtering technique. This model will be introduced in the fourth section of this chapter and discussed more in detail in the remaining chapters. The first four sections of this chapter are aimed at describing Neural Networks in a broader context, they prepare for a complete understanding of the sections that follow.

2.1 Basic Components of Neural Network Models

A Neural Network is specified completely by the definition of its physical parts and the algorithms that are applied to it. Like any other network or graph, a Neural Network consists of processing units that are physically called nodes and serve as the artificial counterpart of neurons in the brain. It also consists of edges defining its topology or connectivity. In the succeeding material, the information-processing algorithm within a Neural Network is completely specified by defining the three following components : the functionality of processing units, topology, propagation and learning rules. with the learning rule being the key to the processing algorithm. Another algorithm, which may require some programming, is also required for any processing system. In Neural Networks, this task is summarized into a simple learning rule.

2.1.1 Processing units

The processing units in a Neural Network mimic the function of neurons in the brain. Therefore, information processed in them is determined by how the input of each node relates to its own output and also how each processing unit connects to others or the environment. This is described completely in these terms [1] by

- Input into the unit from other units or the environment.
- Output sent to other units or the environment.
- The unit's internal state, also referred to as the unit's activation.
- The rule for computing the next state of the unit also called the activation rule or activation function
- The rule for computing the output from the current state. This is defined by the output function.

Biological neurons are complicated physical systems. Their dynamics, if realistically modeled would involve a large number of differential equations. This feature of biological neurons is the origin of a strong motivation toward simplifying the neuronal model. A common means of simplification is to have two separate models each describing a functional mode of the neuron. The two functional modes are 1)the processing mode, and 2)the learning mode.

The dynamics of the *processing state* are described by a single differential equation. Widespread models such as perceptrons own processing units with no genuine state. Their output is a simple function of the input. This function is based primarily on the simplified neuron model of Pitts [9]. The simplest version is a step function of the weighted sum of individual inputs. This function is described in the following equation :

$$f(x,\sigma) = \begin{cases} 1 & \text{if } x > \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

An attempt to make the activation function differentiable resulted in the following so-called logistic function.

$$f(x,\sigma) = \frac{1}{1 + e^{-(x-\sigma)}} \tag{2.2}$$

The processing state is described as combining inputs into a weighted sum, then passing them by the help of a so-called activation function such as the step or sigmoid function.

As described above, the processing state is the equivalent to the state of activation of a simplified artificial neuron. However, the *learning state* is characterized by a vector of parameters of the activation function. These parameters are to be changed according to a desired performance criteria of a Neural Network. The parameters of an activation function consist of

- Weights of individual inputs, with the help of which the weighted sum is computed.
- A threshold of the step or sigmoid function.

2.1.2 Topology

If a system made of the processing units that are described in the previous section is capable of solving complex tasks in a way reminiscent of human cognitive capabilities, its sophistication is attributed to the processing units themselves and more importantly to the way they interact. The topology of the network is what determines the interactions and thus is of crucial importance for the performance of the network.

There are various cases of interconnected networks. The simplest one is a completely interconnected network. This topology allows for arbitrarily complex feedback structures. Although feedback extends the class of possible behaviors immensely, in some cases and for reasons not mentioned, it may be desirable to exclude the possibility of feedback in the topology level. The way to achieve feedback exclusion is simply to make it impossible, following any path along directed connections, to enter a unit once left. Such a network, is called a *feedforward network.*

Since a feedforward network has the properties of a directed graph, each unit in the network can be assigned a rank giving the number of units on the longest path between it and some input unit. It is then easily understood that, for any pair of interconnected units, the rank of the unit the connection leaves is lower than the rank of the unit the connection enters.

The scheme for iterative evaluation according to the rank of individual units suggests for collecting units into subsets of equal rank. Adding the restriction that the rank difference between interconnected units is 1, we obtain a feedforward layered network.

The evaluation of this network is done in a number of iterations that equals the number of interconnection layers. The kth layer consists of the connections between the units with ranks k and k - 1. This evaluation can be viewed as a sequence of vector operators. A simple case of this evaluation is that of a single-layer feedforward

network, which is the topology used for one of the first Neural Networks models, the perceptron of Rosenblatt [10].

We should also mention the other networks that are layered. These are the *feedback layered networks* with some examples like the Adaptive Resonance Theory(ART) of Grossberg [11] and the Bidirectional Associative Memory of Kosko [12].

2.1.3 Propagation rule

After a thorough description of the Neural Network in terms of its processing units and topology, what now remains is to describe the propagation rule for information processing. This rule will specify the time at which, and the order in which, the unit activations are to be updated. The approach that is closest to the biological system is a simultaneous and *asynchronous* update of all units.

Since the asynchronous approach does not seem to have any clear advantages, most existing propagation rules are *synchronous* and vary accordingly with the Neural Network model type. The variation of the propagation rule from a network type to the other is manifested in the two algorithms that we are to describe next.

The first one is for feedforward layered Neural Networks and is a modification of the algorithm used for (FFANN). The algorithm is described in these basic steps:

- The input units are set equal to the input pattern.
- In the kth step, the activation of units of the kth layer are computed using the activations of the $(k - 1)$th layer.
- After a number of steps equal to the number of layers, the state of the output units corresponds to the output pattern.

The processing in this FFANN is viewed as the evaluation of a nested non-recursive function. The situation is different for feedback networks. A detailed discussion of the propagation rule including the steps of the algorithm to be followed in the processing of information in a FFANN are found in [1].

2.1.4 Learning rule

The learning rule is the most difficult constituent of a Neural Network. Formally, it resembles the propagation rule since it also operates on a network of interconnected units. The learning rule consists of changing the state of individual units (the state of activation function parameters) using inputs from the neighboring units. Given the existence of variations of the particular form of this rule, we will limit our discussion of this rule to a feature that is common to many models. This feature is observed in the fact that most learning rules are modifications of stochastic gradient optimization of a certain objective function or performance criterion.

Models that have this feature include the following:

- The error backpropagation algorithm of Rumelhart, Hinton, and Williams [13]
- The perceptron of Rosenblatt [10], minimizing a differentiable modification of the Bayesian misclassification rate.
- The unsupervised learning rule of Oja[14], minimizing the difference between the original pattern and the pattern reconstructed from its feature representation.

Other aspects of learning rules with a classification according to learning types are discussed in the next section.

2.2 Learning Types

The current interest in Artificial Neural Networks methods is mainly attributed to their ability to learn from experience. This ability to learn offers a powerful alternative to programming.

Learning types or methods can be classified as supervised and unsupervised, with a great many paradigms implementing each type.

2.2.1 Supervised learning

Examples of supervised learning paradigms are the original perceptron, and more recently backpropagation. Supervised learning involves the training of the network on a *training set* consisting of vector pairs. One vector is applied to the input of the network; the other is used as a "target" which represents the desired output.

Training is performed by an adjustment of the network weights so as to minimize the difference between the desired and actual output. This process can either be achieved in an iterative procedure, or the weights can be computed by closed form equations. The latter form of training may fail to qualify the system as a Neural Network since its method is so far from the biological method. However, such methods are useful, and provide a broader definition of Artificial Neural Networks.
We can describe iterative training in the following statements:

- Apply an input vector to the network.
- Compare the output produced to the target vector.
- Use the error signal obtained from the above comparison to modify the network weights.

The modification of the weights which we can describe as the correction of the weights can be general, equally applied as a reinforcement to all parts or it may be specific, with each weight receiving an appropriate adjustment. The intention behind adjusting the weights at each step is to reduce the difference between the output and target vectors. Vectors from the training set are applied to the network repeatedly until the error is at an acceptably low value. The training process is successful if the network is capable of performing the desired mapping.

2.2.2 Unsupervised learning

This type of learning, in contrast with supervised learning, requires no information and only needs the input vectors to train the network. Unsupervised learning is also called self-organization [15]. In the course of training, the network weights are adjusted so that similar inputs produce similar outputs. The training algorithm accomplishes this by extracting statistical regularities from the training set, representing them as the values of network weights. As Wasserman phrased it in [2];"self-organization is reminiscent of the manner in which, in some cases, the human brain modifies its structure under the influence of its experiences without a 'teacher' ".

Although applications of unsupervised learning are not as frequent as for supervised learning, they have been used in combination with other paradigms and have produced useful results. An example of these applications is the counterpropagation method[16].

2.3 Computational Features of Neural Networks

Neural networks research is directed toward finding models that can accomplish useful tasks such as association, pattern recognition etc. The most important classes Of *application tasks* that Neural Networks are able to perform will be identified in this section.

2.3.1 Function approximation

A deterministic feedforward network represents a mapping between the input layer units and the output layer units. The evaluation of the mapping is achieved by propagating the activations from the input layer to the output layer.

The network topology and activation functions are what determine the class of

functions that can be represented by a network. For instance, a single-layer perceptron output results from the input in the following way:

$$y = \sum_i w_i x_i \tag{2.3}$$

It is then clear that the class of linear mappings can be represented by the linear perceptron. However, a broader range of function classes, usually nonlinear mappings, have been represented by multi-layer perceptrons. In addition to single-layer and multi-layer perceptrons, layered feedforward Neural Networks with radial basis activation functions seem to perform especially well for general functional approximation tasks. One of the functional approximation tasks that is of great application importance is classification. Even though any network model could be used for classification, specific properties of this application require particular activation functions and learning rules.

The classification task is characterized in the following way:

- A set of objects that are each characterized by a fixed-length vector of numeric or Boolean features is supplied. A set of classes is also given, such that each object is only assigned to one class.
- A subset is then selected and called a *training set*, for which the class assignment is explicitly known.
- The Neural Network can be *trained* to estimate the class of objects that did not belong to the training set.
- The performance of a neural classifier is measured by the proportion of objects for which their class assignment estimate (recognition rate) is correct.

The feature vector describing the objects is frequently referred to as a *pattern* and the entire task as *pattern recognition.*

2.3.2 Data compression

The task of a Neural Network model applied to data compression is to find a mapping that reduces the original pattern to a compressed pattern, usually of a substantially lower dimension. The learning rule of Oja[14] is a Neural Network model for linear data compression. However, for nonlinear data compression, multilayer perceptrons in an autoassociative mode can be used: The desired output of the perceptron is set equal to the input. If an autoassociative perceptron with a hidden layer that is narrower than the input is used (and output) layer is trained to produce outputs that are very close to the inputs, the hidden layer constitutes a nonlinear compression of input patterns. The inverse mapping of compressed patterns to the original ones is given by the output layer.

2.3.3 Optimization

The class of optimization tasks that can be solved by Neural Networks are of the type that can be solved by relaxation of annealing and can be characterized by the following properties:

- *A neighborhood* system must be defined on the variables of the optimization task.
- The objective function to be maximized has to be *additive* in terms of *cliques*, groups of variables within which each variable is a neighbor of every other variable.

As in the case of functional approximation, optimization consists of a number of subclasses which can also be viewed as task classes. One of the subclasses that is frequently tackled by Neural Networks approaches is associative memory.

2.4 Examples of Models of Feedforward Neural Networks

Given that the proposed research in this document will be restricted to FFANN in a supervised learning framework, only the models that belong to the same setting will be reviewed in this section. The discussion of additional models can be found in [1].

2.4.1 Perceptron and adaline

The perceptron model proposed by Rosenblatt [10] is one of the first models ever proposed and is still important to current research. The classification of visual patterns was one of its primary goals. Widrow [17, 18] also has pursed a similar approach. The perceptron is the simplest version of a (FFANN). It consists of a single layer for input units and a single output unit. The one layer inputs are weighed by vector of connection strengths then fed into a step function. The activation function of the output unit is described as follows:

$$z = \delta(\sum_i w_i x_i - \sigma) \tag{2.4}$$

$$\delta(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

with x_i being the ith element of input patterns, z the output, w_i the connection weight of the ith input, and σ the output unit's threshold. The output unit activation can assume two values, zero and 1. Each value corresponds to one of two pattern classes that have to be separated.

The learning rule of the perceptron is supervised . It consists of a very simple strategy of changing the weights only if the pattern $\mathbf{x}$ is misclassified, that is, if the activation of the output unit for this pattern is not equal to the correct class of the pattern. The amount of change is

$$\Delta w_i = \begin{cases} x_i & \text{if class} = 1 \\ -x_i & \text{if class} = 0. \end{cases} \tag{2.6}$$

This simple learning rule has an important property. If both classes are linearly separable, the weights will converge to the values that materialize the separation.

2.4.2 Multilayer perceptron

The fact that single-layer perceptrons can only separate linearly separable classes was pointed out by Minsky and Papert [110] at the end of the 1960's, and this led to a substantial decrease in interest in Neural Networks research.

Rumelhart along with Hinton and Williams [13] formulated the *backpropagation learning rule.* The backpropagation model is based on two principles:

- 1. To overcome the limitations of the single-layer perceptron, all that is necessary is to insert one or more additional layers between input and output. These layers consist of processing units with nonlinear activation functions, typically sigmoid functions (3). Arbitrary convex classes can be separated by a network with one such hidden layer (a two-layer network), and arbitrary non convex classes by a network with two hidden layers (a three-layer network);

- 2. The delta rule (8) can be used to learn output layer weights. Remaining weights can be learned by recursive application of the chain rule for computing derivatives. For a two-layer network with x_i representing input unit activations, y_j hidden-unit activations, z_k output unit activations, w_{kj} output layer weights, and v_{ji} hidden-layer weights, the rule is the following:

$$\begin{aligned} \frac{\partial E}{\partial v_{ji}} &= \sum_k \sum_j \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial y_j} \frac{\partial y_j}{\partial \sum_i v_{ji} x_i} \frac{\partial \sum_i v_{ji} x_i}{\partial v_{ji}} \\ &= \sum_k \sum_j (z_k - d_k) w_{kj} y_j (1 - y_j) x_i \end{aligned} \tag{2.7}$$

The related recursive formulas are given in [13]

A large number of applications are based on the model constituted by the simple principles described above.

2.5 The Optimal Filtering Technique

The computation of the weights of a FFANN in order to achieve an input/ output mapping is a problem that could be classified as a parameter estimation problem or a high dimensional nonlinear system identification problem. Nonlinear optimization techniques are well suited to solve the problem. The only disadvantage of these techniques is their computational cost especially if the problem is attacked globally instead of being divided into a set of manageable subproblems. The problem of estimating the weights in a (FFANN) is nonlinear due to the sigmoidal activation function, but it is smooth. Therefore methods of linear estimation theory could be applied to such a nonlinear problem by linear approximation of the effects of small perturbations in the state of the nonlinear system from a "nominal" value. Since the state variables are not known beforehand the nominal trajectory has to be defined "on the fly"[19] as the current best estimate of the actual trajectory. The approach used here is called *extended kalman filtering.* The advantage in this approach is that the perturbations include only the state estimation errors which are usually smaller than perturbations from any predefined nominal trajectory and therefore better conditioned for linear approximation.

2.6 Thesis Problem Description

During the investigation of the work done so far in using the kalman filter algorithm in the supervised training of the (FFANN), the major problem that attracted our

attention is that several trials and simulations had to be done before the performance was satisfactory. The need for several trials, as we understand from the work previously reported, is that for certain initial conditions the algorithm was trapped in local minima that were not acceptable in terms of performance. The only way to escape the problem that has been adopted so far is to run the algorithm with different initial conditions. No major work has been done to make the algorithm escape the local minima once it is trapped. Even though the algorithm has the advantage of being stable under certain conditions, it does not always converge to a satisfactory solution. Our main goal in this thesis is to develop solutions for the problem outlined above.

2.7 The Objectives and the Outline of the Thesis

Having highlighted the problems that are of interest to us in the previous section, we now, address these problems by stating the goals that we plan to achieve. A research plan is then developed that includes the tasks to be accomplished.

As mentioned earlier in this document, guesswork due to tunable parameters, etc. can hurt the efficiency of training a Neural Network to perform a desired task. This is one of the factors that has motivated researchers to focus their efforts on seeking algorithms that are more suitable, easily comprehended and also theoretically supported. It is believed that better understanding of an algorithm and the main theory behind it, is the basis for efficient training of (FFANN). The goal of two planned research tasks in this work is to improve the efficiency of a newly introduced feedforward training algorithm without a degradation of its performance.

2.7.1 Outline of the thesis

Since the research goals have already been determined, the research plan will be organized in a manner as to achieve these goals and arrive at a stage of subsequent developmental research. The tasks to be accomplished in the proposed work include the following main ones:

Task No 1: *Characterize the essential features of the learning algorithms based on Kalman filters.*

Objective: Through an investigation of the theory behind the algorithm and computer simulations to observe the behavior, point out the problems that this algorithm faces in terms of how the convergence is affected with different initial conditions. It is also beneficial to be aware of the assumptions made, if any, while designing this algorithm or any post-improvement of it.

Significance: The simulation of algorithms that are theoretically comprehended and convergent will be meaningful in two ways. On one hand, it will help identify any problems with the algorithm that were or were not theoretically anticipated. On the other hand, it will provide insight or how to modify the algorithm to work better in terms of computation and convergence time efficiency.

Approach: We start with a careful review of the work that was done on the subject until present. This is accomplished by gathering most of the literature published. The theoretical understanding of the work that was previously published is the only way to discover any discrepancies or neglected issues in the problem. The next step is to check the accuracy of the results by a careful simulation. Also, a verification of any claims that are not theoretically supported will be conducted during the computer simulation. This way we can discover

all the weaknesses of the method or the algorithm, and by further investigation, a modification will be developed for future improvements.

Task No 2: *Develop a modification of the learning algorithm aimed at its improvement.*

Objective: To overcome the identified problems of the previously proposed algorithm(see appendix); the major ones being convergence to none satisfactory solutions and the high load of computation.

Significance: Once the modification is successful, this algorithm will be very competitive with the backpropagation algorithm, especially in terms of the computational efficiency.

Approach: Having identified the problems with the previous approach to the problem at hand, a modification is suggested based on the following two criteria:

- The modification should avoid any approximations or assumptions that were not theoretically justified.
- The modification is aimed toward enabling the algorithm to converge to satisfactory solutions regardless of the choice of initial conditions and in an adequate amount of time.

After the development of the modification that is based on theoretical understanding and aims toward an improvement of the algorithm , simulations on different examples are conducted. The results are then compared with the previously reported ones. The simulation results are expected to support what was anticipated from the modification theoretically.

CHAPTER 3

The Extended Kalman Filter Algorithm

3.1 Motivation

The classical (discrete-time) backpropagation algorithm, although widely used, has numerous disadvantages. One of them is an extreme sensitivity to the initial choice of the set of weights: There are many choices of weights for which the algorithm will not converge and finally when it does converge for some set of weights, the error diminishes only after a great number of iterations. There is also another problem with the backpropagation algorithm exhibited as an inconsistency in training. This means, the mean squared error could remain the unchanged for many iterations then suddenly decrease to a lower value.

The Extended Kalman Filter algorithm, presented here, does not suffer from such problems, it is not extremely sensitive to the initial choice of weights and it converges comparatively faster.

3.2 The Learning Dynamics

The task of finding weights, which would achieve a specific input/output mapping, is viewed as a parameter estimation problem. The network we use is a fully connected Feedforward Artificial Neural Network or (FFANN) which is multi-dimensional and nonlinear. The parameters to be estimated are the weights [20, 21, 8, 22, 23, 24]. An arbitrary FFANN which depends on the application is considered. It consists of a large T dimensional weight vector θ made up of all the synaptic connections with sigmoidal nonlinearities. The objective in training the (FFANN) is to determine the weight values producing the L dimensional output vectors $g(n)$ which are the closest possible to the desired output values $d(n)$. The input sequence $i(n)$ is made up of N dimensional vectors.

Here, we consider searching for the weights *on-line* with patterns presented in sequence at the input of the network and with updates made recursively. The synaptic weight vector θ is viewed as the state of a static nonlinear dynamical system described by the equations

$$\theta(j) \;\; = \theta(j-1) \;\; = \theta_0$$

$$\mathbf{d}(j) \;\; = \;\; \mathbf{h}(\theta_0, \mathbf{i}(j)) + \mathbf{e}(j),$$

where j is the time index, $\mathbf{h}(\theta_0, \mathbf{i}(j)$ is the time varying function describing the network and $\mathbf{e}(j)$ is the multidimensinal sequence of modeling errors[21]. To obtain the dynamic estimates $\hat{\theta}(n)$ of θ_0, a traditional approach is adopted based on a deterministic formulation, in which $(\mathbf{i}(j), \mathbf{d}(j))$ is a sequence of non-random input/output pairs and the cost to be minimized at time n is

$$\epsilon(\hat{\theta}(n)) = \sum_{j=1}^{n} \| \mathbf{d}(j) - \mathbf{h}(\hat{\theta}(n), \mathbf{i}(j)) \|_2 \, \lambda^{n-j}. \tag{3.1}$$

Here $\mathbf{e}(j)$ is the deterministic sequence of the modelling errors, and λ^{n-j} $(0 < \lambda < 1)$ is a forgetting function which exponentially discounts the examples presented in the past. This is the so-called *weighted recursive least squares approach*(WRLS). In the case of **h** being linear, the solution could be obtained using the classical RLS algorithm [8].

3.2.1 The global extended Kalman filter algorithm

The extended Kalman filter(EKF) equations are derived by expanding the nonlinear function h(θ,i(n)) around the current estimate parameter vector $\hat{\theta}(n$ - 1)(estimated from all the data up to time (n - 1)). Namely, the state model is rewritten as

$$\theta(n) \;\; = \theta(n-1) \;\; = \theta_0$$

$$\mathbf{d}(n) \;\; = \;\; \mathbf{h}(\hat{\theta}(n-1), \mathbf{i}(n)) + \mathbf{H}^T(n)(\theta_0 - \hat{\theta}(n-1)) + \rho(n) + \mathbf{e}(n), \qquad (3.2)$$

where $\mathbf{H}(n)$ is the T matrix given by

$$\mathbf{H}(n) = \frac{\partial \mathbf{h}(\theta, \mathbf{i}(n))}{\partial \theta} \mid_{\theta=\dot{\theta}_{(n-1)}} \qquad (3.3)$$

and $\rho(n)$ is the residual in the Taylor expansion of **h**.

The state model becomes

$$\theta(n) \;\; = \theta(n-1) \;\; = \theta_0$$

$$\mathbf{d}(n) \;\; = \mathbf{H}^T(n)\theta_0 \;\; + \xi(n) + \mathbf{e}(n), \qquad (3.4)$$

with $\xi(n) = \mathbf{h}(\theta(n-1)), \mathbf{i}(n))$ - $\mathbf{H}^T(n)\hat{\theta}(n-1) + \rho(n)$.

Both, in the deterministic and in the stochastic formulations the estimate $\hat{\theta}(n)$ is obtained as the optimal regression of θ_0 in (3.2). We concentrate on the deterministic formulation, for which the estimates are obtained from the minimization of

$$\epsilon(n) = \sum_{j=1}^{n} \| \mathbf{e}(j) \|_2 \, \lambda^{n-j}. \tag{3.5}$$

The solution is derived from the normal equation

$$\nabla_{\theta_0} \epsilon(n) \quad = 2 \textstyle\sum_{j=1}^{n} \mathbf{H}(j)(\mathbf{d}(j) - \mathbf{H}^T(j)\theta_0 - \xi(j))\lambda^{n-j} \quad = 0. \tag{3.6}$$

which gives

$$\hat{\theta}(n) = \Phi^{-1}(n)\mathbf{r}(n), \tag{3.7}$$

with

$$\Phi(n) = \sum_{j=1}^{n} \mathbf{H}(j)\mathbf{H}^T(j)\lambda^{n-j}, \tag{3.8}$$

$$\mathbf{r}(n) = \sum_{j=1}^{n} \lambda^{n-j}\mathbf{H}(j)(\mathbf{d}(j) - \xi(j)) \tag{3.9}$$

The key assumption here is that that the residual sequence $\xi(n)$ does not depend on θ_0.

The EKF recursions for the network are

$$\mathbf{G}(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{H}(n) \, [\mathbf{I} + \lambda^{-1}\mathbf{H}^T(n)\mathbf{P}(n-1)\mathbf{H}(n)]^{-1}, \tag{3.10}$$

$$\hat{\theta}(n) = \hat{\theta}(n-1) + \mathbf{G}(n)(\mathbf{d}(n) - \mathbf{h}(\hat{\theta}(n-1), \mathbf{i}(n)), \tag{3.11}$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{G}(n)\mathbf{H}^T(n)\mathbf{P}(n-1). \tag{3.12}$$

$\mathbf{G}$ is a T matrix (the so called Kalman gain) and $\mathbf{P}(n) = \Phi^{-1}(n)$ is a T matrix.

To initialize the algorithm, the covariance matrix is chosen as $\mathbf{P}(0) = \delta^{-1}\mathbf{I}$, with δ

greater than zero, a small arbitrary value and weights $\hat{\theta}(0)$ to some nonzero random values.

3.2.2 Computation of the gradient matrix

In this section we take a closer look to the gradient matrix $H^T(n)$. Given the architecture of the (FFANN), the elements of the weight vector θ are arranged so that the rows of $H(n)$ contain the derivatives of the global outputs with respect to all the T synaptic weights for $\theta = \hat{\theta}(n-1)$. The neurons are numbered from 1 to η_s, so that $\theta = (w_1, w_2, ..., w_{\eta_s})$. The equation describing the desired output can then be rewritten as :

$$\begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_L \end{pmatrix}(n) = \begin{pmatrix} (\frac{\partial g_1}{\partial w_1})^T & (\frac{\partial g_1}{\partial w_2})^T & \cdots & (\frac{\partial g_1}{\partial w_{\eta_s}})^T \\ (\frac{\partial g_2}{\partial w_1})^T & (\frac{\partial g_2}{\partial w_2})^T & \cdots & (\frac{\partial g_2}{\partial w_{\eta_s}})^T \\ \vdots & \vdots & \vdots & \vdots \\ (\frac{\partial g_L}{\partial w_1})^T & (\frac{\partial g_L}{\partial w_2})^T & \cdots & (\frac{\partial g_L}{\partial w_{\eta_s}})^T \end{pmatrix} \times \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_{\eta_s} \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_L \end{pmatrix}(n)$$

Where w_i is the vector containing the synaptic weights of neuron i, and $\epsilon(n) = \zeta(n)+e(n)$. The computation of the derivatives in the matrix $\mathbf{H}(n)$ is achieved via a back propagation of the output values $\mathbf{g}(n)$ through the network.

In order to illustrate how these derivatives are obtained, we first need to describe the structure of the (FFANN) used. The description included in this section is identical to the one given in [25] since their description of the structure was clear and complete.

Let M be the total number of layers in the (FFANN) with the input and output layers included. The $i-th$ neuron in the $s-th$ layer is denoted by neuron (s,i), and η_s denotes the total number of neurons in the $s-th$ layer. x_i is the input of the neuron $(1,i)$, x_i^s is the output of the neuron (s,i), $w_{i,k}^s$ is the linkweight coefficient

from the neuron (s,k) to neuron $(s+1,i)$, and $\hat{w}_i^s$ is the threshold of the neuron (s,i). Usually, the thresholds are treated as weights that connect an input unit, which os always on, i.e. its value is always one, to the neuron, then

$$x^{s-1}_{\eta_{s-1}+1} = 1,\ 2 \leq s \leq M \quad \text{and} \quad w^{s-1}_{i,\eta_{s-1}+1} = \begin{cases} \hat{w}_i^s & \text{if } 2 < s < M \\ 0 & \text{if } s = M \end{cases}$$

Further, let

$$z^s = \begin{pmatrix} z_1^s \\ \vdots \\ z^s_{\eta_s} \end{pmatrix},\ x^s = \begin{pmatrix} x_1^s \\ \vdots \\ x^s_{\eta s+1} \end{pmatrix},\ w_i^{s-1} = \begin{pmatrix} w^{s-1}_{i,1} \\ \vdots \\ w^{s-1}_{i,\eta_{s-1}+1} \end{pmatrix},\ w^{s-1} = \begin{pmatrix} w_1^{s-1})^T \\ \vdots \\ w^{s-1}_{\eta_s})^T \end{pmatrix}$$

where w_i^{s-1} is a $(\eta_{s-1}+1) \times 1$ weight vector of the neuron(s,i), w^{s-1} is a $\eta_s \times (\eta_{s-1}+1)$ weight matrix of the $s-th$ layer, and x^s is a $\eta_s \times 1$ output vector of the $s-th$ layer. Then, the operation equation of the network can be expressed in the following vector form

$$z^s = \begin{cases} x & \text{if } s = 1 \\ w^{s-1}x^{s-1} & \text{if } 2 \leq s \leq M \end{cases} \quad \text{and } x^s = \begin{cases} z^s & \text{if } s = 1 \text{ or } M \\ h(w^{s-1}x^{s-1}) & \text{if } 2 \leq s \leq M-1 \end{cases}$$

where the function $h(.)$ may be chosen as the hyperbolic tangent function $h(x) = \tanh(x)$ and

$$h(w^{s-1}x^{s-1}) = \begin{pmatrix} h((w_1^{s-1})^T x^{s-1}) \\ \vdots \\ h((w^{s-1}_{\eta_s})^T x^{s-1}) \end{pmatrix}$$

The weight training algorithm requires the computation of the partial derivatives of the output x^M of the (FFANN) with respect to the weight vectors. These derivatives may be derived from the operation equation stated above. A computation procedure for partial derivatives from layer $(M-1)$ to layer $(M-2),\cdots$, until the layer 2 of the (FFANN) is considered. These derivatives are given as follows

$$\frac{\partial x^M}{\partial w_\beta^{M-1}} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (x^{M-1T} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where $\frac{\partial x^M}{\partial w_\beta^{M-1}}$ are $(\eta_{M-1}+1)\times\eta_M$ matrices. For $2 \leq i \leq M-1$, $\beta = 1, 2, \cdots, \eta_{M-i+1}$; one can obtain

$$\frac{\partial x^M}{\partial w_\beta^{M-i}} = \frac{\partial x^M}{\partial x^{M-1}} \frac{\partial x^{M-1}}{\partial x^{M-2}} \cdots \frac{\partial x^{M-i+2}}{\partial x^{M-i+1}} \frac{\partial x^{M-i+1}}{\partial w_\beta^{M-1}} = (\prod_{j=0}^{i-2} \frac{\partial x^{M-j}}{\partial x^{M-j-1}}) \frac{\partial x_{M-i+1}}{\partial w_\beta^{M-i}}$$

Also for $2 \leq i \leq M-1$, $\beta = 1, 2, \cdots, \eta_{M-i+1}$, the following relations are obtained from the operation equation

$$\frac{\partial x^{M-j}}{\partial x^{M-j-1}} = \begin{cases} w^{M-1} & \text{if } j = 0 \\ \begin{pmatrix} h'((w_1^{M-j-1})^T x^{M-j-1}) w_1^{M-j-1} \\ \vdots \\ h'((w_{\eta_{M-j}}^{M-j-1})^T x^{M-j-1}) w_{\eta_{M-j}}^{M-j-1} \end{pmatrix} & \text{if } j \neq 0 \end{cases}$$

$$\frac{\partial x^{M-i+1}}{\partial w_\beta^{M-i}} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ h'((w_\beta^{M-i})^T x^{M-i}) x^{M-i} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where $h'(x) = 1/\cosh^2(x)$.

3.2.3 The local appraches and the (NDEKF)

Due to the high computational load of the EKF algorithm, researches thought to simplify the global approach by partitioning the global problem into a set of manageable subproblems. In the case of (FFANN), the partition of the problem may be down to a group of neurons, to the layer level, to the level of the single neuron, or, ultimately down to the level of a single synaptic weight. In this work, we are interested in partitioning the problem down to the neuron level and solve the global problem(minimizing the cost) by independently updating the weights *each neuron* at each step.

The effect of the neuron i in the global output of the network is locally described by the gradient matrix $H_i^T(n)$ [26]which is the $i-th$ column of the $H^T(n)$.

$$\mathbf{H}_i^T(n) = \begin{pmatrix} (\frac{\partial g_1}{\partial w_i})^T & (\frac{\partial x_1^M}{\partial w_i})^T \\ (\frac{\partial g_2}{\partial w_i})^T & (\frac{\partial x_2^M}{\partial w_i})^T \\ \vdots & \vdots \\ (\frac{\partial g_L}{\partial w_i})^T & (\frac{\partial x_L^M}{\partial w_i})^T \end{pmatrix}$$

Since the gradient matrix is now decomposed into a set of smaller gradient matrices, we can also think abot decomposing the weight vector into few subvectors each one associated with the corresponding gradient matrix.

The decomposed parameter estimation problem is described by the following equation:

$$y(k) = f(w, x(k)) = f(w_1^1, \cdots, w_{\eta_2}^1, \cdots, w_1^{M-1}, \cdots, w_{\eta_M}^{M-1}, x(k))$$

where $y(k)$ is a $\eta_M \times 1$ vector of output of the network at time k, w_β^α, $1 \leq \beta \leq \eta_{\alpha+1}$, $1 \leq \alpha \leq M-1$ are $\eta_\alpha \times 1$ weight vectors to be estimated, $x(k)$ is a $\eta_1 \times 1$ vector of the input of the network at time k, $f(.)$ is a $\eta_M \times 1$ smooth vector function.

We now introduce the new parameter vectors θ_i, $1 \leq i \leq n$, $n = \sum_{l=2}^{M} \eta_l$ that

represent the weight vectors w_β^α, let $\theta_i = w_\beta^\alpha$, then the scripts i and α, β are related by $\alpha_i = 1$, $\beta(i) = i$, if $i \leq \eta_2$ and $\alpha(i) = 1 + \max\{j : (i - \sum_{l-2}^{j} \eta_l) > 0\}$, $\beta(i) = i - \sum_{l-2}^{\alpha-1} \eta_l$, if $i > \eta_2$. Consequently the θ_i are $(\eta_{\alpha(i)+1}) \times 1)$ vectors, and the nonlinear euation is rewritten as $y(k) = f(\theta_1, \cdots, \theta_n, x(k))$. Given that $x(k)$ is the input pattern, $y_d(k)$ is a $\eta_M \times 1$ vector of desired output pattern of the (FFANN), and θ_i at time k. The weight training oof the (FFANN) is aimed toward estimating the weight vectors θ_i such that the output $y(k)$ of the FFANN tracks the desired output $y_d(k)$ with an error that the algorithm can make converge to zero as $k \longrightarrow$ inf.

In order to decouple the global problem acuretly, we can only use the $(M_{i+1} \times M_{i+1})$ diagonal blocks of the error covariance matrix. Neglecting the error covariance matrices $p_{ij}(k)$ of the estimations $\theta_i(k)$ in the recursive estimation procedure is based on the assumption that the error covariance matrix $\mathbf{P}(n)$ normally contains most of its energy around its diagonal blocks. [27]. It also results in obtaining an approximate *neuron-decoupled* extended kalman filter equations or NDEKF formulations from the standard extended kalman filter (EKF) formulations [25] as follows:

$$A(n) = [I + \lambda^{-1} \sum_{j=1}^{n} H_j p_j(n-1) H_j^T(n)]^{-1}, \tag{3.13}$$

$$G_i(n) = \lambda^{-1} p_i(n-1) H_i^T(n) A(n), \tag{3.14}$$

$$p_i(n) = \lambda^{-1}(I - G_i(n)) H_i(n) p_i(n-1), \tag{3.15}$$

$$\theta_i(n) = \theta_i(n-1) + G_i(n)\epsilon(n), \tag{3.16}$$

$$H_i(n) = \frac{\partial f(\theta_1(n-1), ..., \theta_k(n-1), x(n))}{\partial \theta_i(n-1)}, \tag{3.17}$$

$$\epsilon(n) = y_d(n) - f(\theta_1(n-1), ..., \theta_k(n-1), x(n)), \tag{3.18}$$

where $A(k)$ is a $\eta_M \times \eta_M$ matrix, $G_i(k)$ are $(\eta_{\alpha(i)+1}) \times \eta_M$matrices of the filtering gain, and $p_i(k) = p_i^T(k)$ are $(\eta_{\alpha(i)+1}) \times (\eta_{\alpha(i)+1})$ matrices of the error covariance matrix of the estimation $\theta_i(k)$. The procedure of recursive training described in the above

equations can be explained as follows.The NDEKF uses the old and the new outputs to recursively train the weights. In every iteration, the NDEKF predicts the network output $y(k)$ based on the previous estimated weight vectors and the new input. It then compares the difference between the predicted output $y(k)$ and the new desired output $y_d(k)$, the prediction error $\epsilon(k)$ can be obtained. Based on this prediction error and the information of the entire history of the input, desired output pair, which is stored in the covariance matrices $p_i(k-1)$, a set of modification coefficients, gain matrices $G_i(k)$ corresponding to the weight vectors θ_i of the $neuron(\alpha(i)+1, \beta(i))$, can be calculated. The new trained set of the weight vectors $\theta_i(k)$ is then determined by the sum of the last trained weight vectors $\theta_i(k-1)$ and the innovation term which is a multiplication of the gain matrices $G_i(k)$ and the prediction error $\epsilon(k)$. At last, the information of the new input and desired output is stored in the error covariance matrices $p_i(k)$ to be used in the next iteration.

3.3 Simulation Examples

Simulations, were conducted to compare the performance of the NDEKF to the proposed modifications. In this chapter, we report the performance of the original NDEKF on two continuous 2-dimensional patterns. The XOR problem is described as a 2-region classification problem and is performed on a one of the 2-dimensional patterns. A 4-region classification problem is performed on the second 2-dimensional pattern. The network architecture consists of two inputs followed by two hidden layers. The size of the output layer we use is 2 or 4 depending on the problem. The size of the hidden layers is 10. The network size is thus (2-10-10-2) for the 2-region classification problem and (2-10-10-4) for the 4-region classification problem. This network size has a weight vector, including the bias weights, of 162(respectively 184) elements including the bias weights.

The input pattern was the uniformly distributed random points in the region $[0,1]^2$. The function to be approximated or the desired output values for the network are set to 1 and 0 inside and outside the regions depicted in the figures below. If a point falls into a region of a certain color, shade or class, it is assigned a target output vector for that class. The target output vectors are the two columns of the 2×2 matrix for the 2-region classification problem and the four columns of the 4×4 identity matrix for the four region classification problem. The inputs were not presented in sweeps but obtained continuously by a random number generator. The initial weights were randomly selected and also uniformly distributed between -0.5 and 0.5. The P matrices were initialized to 1000 times the identity matrix. The adaptation of the weights is done as follows:

1. Pick a random point and propagate forward to the output layer.

2. backpropagate the error values (error is computed as the difference between the target output value and the actual computed output) through the whole

network.

3. Simultaneously update all the network weights and matrices.

The mean squared error is averaged every 250 iterations (We call this average: an interval average error).

Simulations for the original (NDEKF) were performed on 5 different initial conditions. The training is completed when a stoppage criterion is achieved. The stoppage criterion is described by a threshold of the interval average error or a maximum number of iterations set by the user. In our simulations we chose a threshold error(0.05, 0.3) respectively. The maximum number of iterations was set to 50,000 for the 4-region classification problem and 25000 for the 2-region classification problem. The performance results are described in the following table and figures. These include results for both the XOR 2-region classification problem and the 4-region classification problem.

We define the Total Average Error (TAE) as

$$\epsilon(n) = (\sum_{j=1}^{n} \| \mathbf{e}(j) \|_2)/n, \tag{3.19}$$

and the Interval Average Error (IAE) as

$$\epsilon_k(n) = (\sum_{j=k}^{k+L} \| \mathbf{e}(j) \|_2)/L, \tag{3.20}$$

where $k = j/L$ when j is a multiple of L.

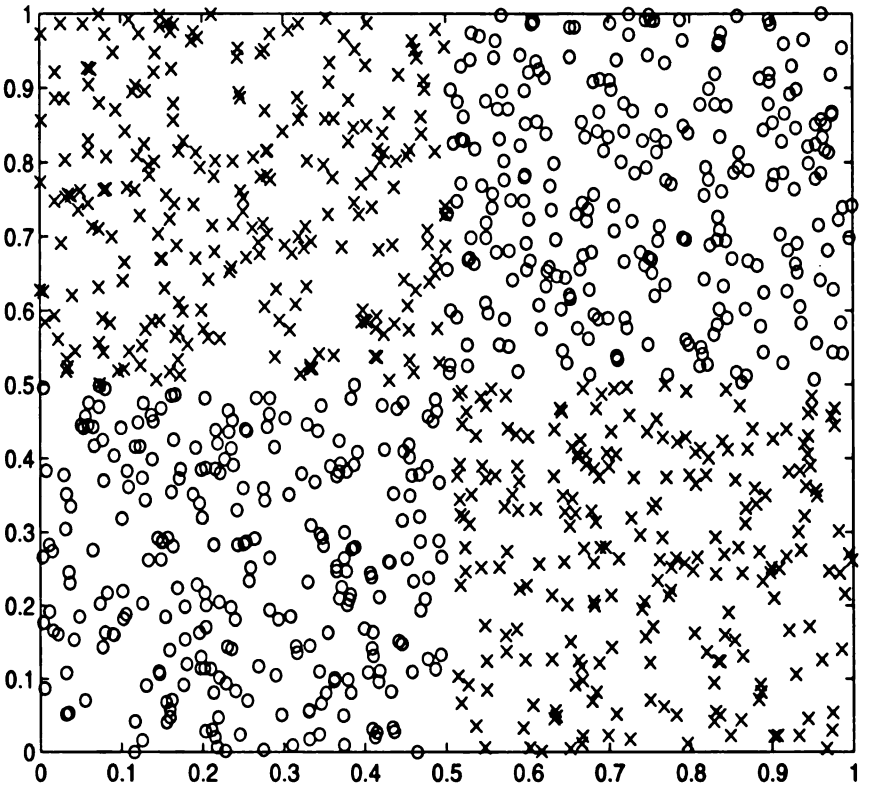


Figure 3.1. The 2-region classification pattern

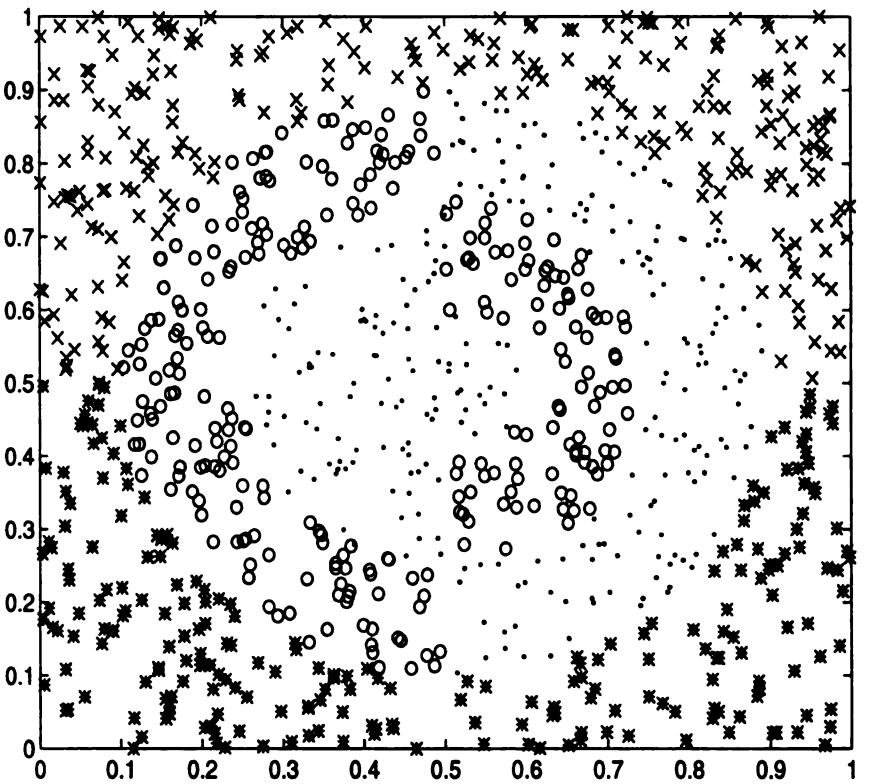


Figure 3.2. The 4-region classification pattern

Table 3.1. The 2-region classification problem

lambda choices		IC1	IC2	IC3	IC4	IC5	IC6
$\lambda = 1$	TAE	0.1095	0.1255	0.1102	0.1326	0.1136	0.1348
	IAE	0.0823	0.1069	0.1022	0.1061	0.1006	0.1176
	class %	97.2	96.2	97.0	96.2	97.0	95.8
	#iterations	25000	25000	25000	25000	25000	25000
$\lambda = 0.99$	TAE	0.7845	1605	0.6530	0.9377	0.7760	0.7759
	IAE	0.9467	17.6	0.5663	1.0554	0.5952	0.9878
	class %	37.3	25.4	80.5	20.2	54	36.9
	#iterations	25000	25000	25000	25000	25000	25000
$\lambda = 0.98$	TAE	NAN	NAN	NAN	NAN	NAN	NAN
	IAE	NAN	NAN	NAN	NAN	NAN	NAN
	class %	0	0	0	0	0	0
	#iterations	5000	5500	3250	3000	3750	2500
$\lambda = 1.01$	TAE	0.2384	0.2597	0.3620	0.3551	0.2706	0.3480
	IAE	0.2452	0.2721	0.3595	0.3614	0.2812	0.3458
	class %	85.7	86.1	77.4	80.0	85.1	79.9
	#iterations	25000	25000	25000	25000	25000	25000
$\lambda = 1.02$	TAE	0.3801	0.5089	0.4659	0.4324	0.5248	0.3724
	IAE	0.3824	0.5565	0.4950	0.4150	0.5255	0.3950
	class %	76.6	69.2	74.1	73.8	65.8	78.1
	#iterations	25000	25000	25000	25000	25000	25000

Table 3.2. The 4-region classification problem

lambda choices		IC1	IC2	IC3	IC4	IC5	IC6
$\lambda = 1$	TAE	0.5757	0.5885	0.5718	0.5357	0.5596	0.5624
	IAE	0.5736	0.5416	0.5295	0.4916	0.5145	0.5210
	class %	76.9	75.1	77.7	77.1	78.2	78.4
	#iterations	50000	50000	50000	50000	50000	50000
$\lambda = 0.99$	TAE	0.7604	4.7649E6	0.6638E3	NAN	NAN	NAN
	IAE	0.7936	0.0179E6	1.5E3	NAN	NAN	NAN
	class %	44.7	22.5	23.3	0	0	0
	#iterations	50000	50000	50000	46750		
$\lambda = 0.98$	TAE	NAN	NAN	NAN	NAN	NAN	NAN
	IAE	NAN	NAN	NAN	NAN	NAN	NAN
	class %	0	0	0	0	0	0
	#iterations	36260	36500	22750	11000	17500	24000
$\lambda = 1.01$	TAE	0.7167	0.7223	0.7002	0.6387	0.7160	0.6636
	IAE	0.7199	0.7030	0.6786	0.6165	0.6958	0.6416
	class %	51.4	51.12	52.5	66.8	51.0	54.0
	#iterations	50000	50000	50000	50000	50000	50000
$\lambda = 1.02$	TAE	0.7375	0.7206	0.7331	0.7046	0.7493	0.6933
	IAE	0.7392	0.6993	0.7331	0.7046	0.7337	0.6687
	class %	49.5	54.2	51.3	54.5	50.4	54.8
	#iterations	50000	50000	50000	50000	50000	50000

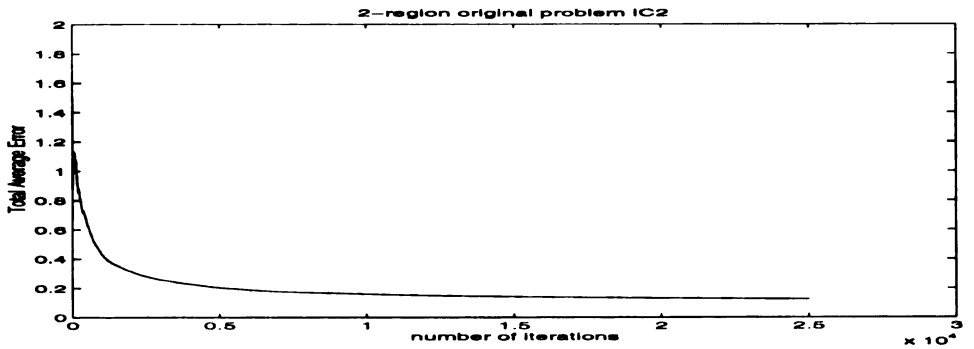


Figure 3.3. The TAE with $\lambda^{-1} = 1$

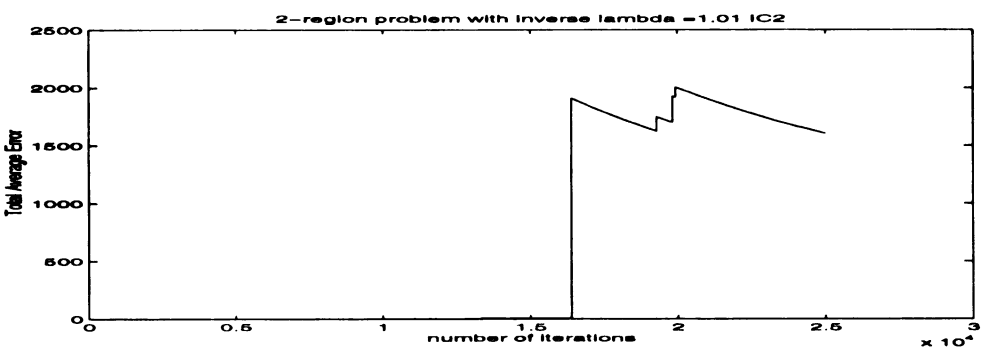


Figure 3.4. The TAE with $\lambda^{-1} = 1.01$

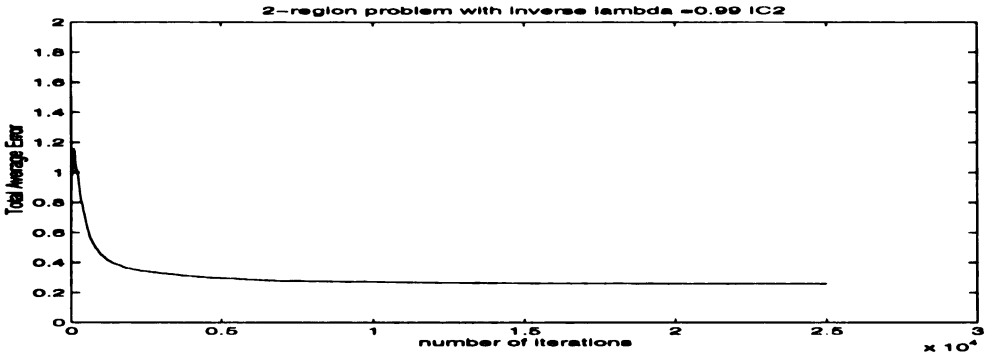


Figure 3.5. The TAE with $\lambda^{-1} = 0.99$

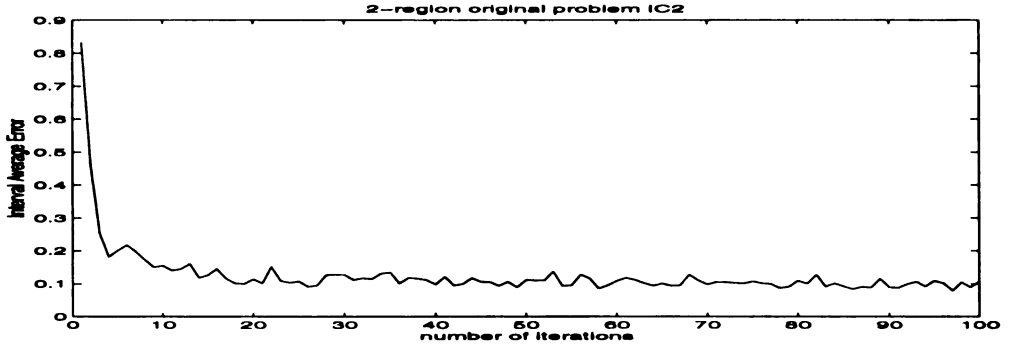


Figure 3.6. The IAE with $\lambda^{-1} = 1$

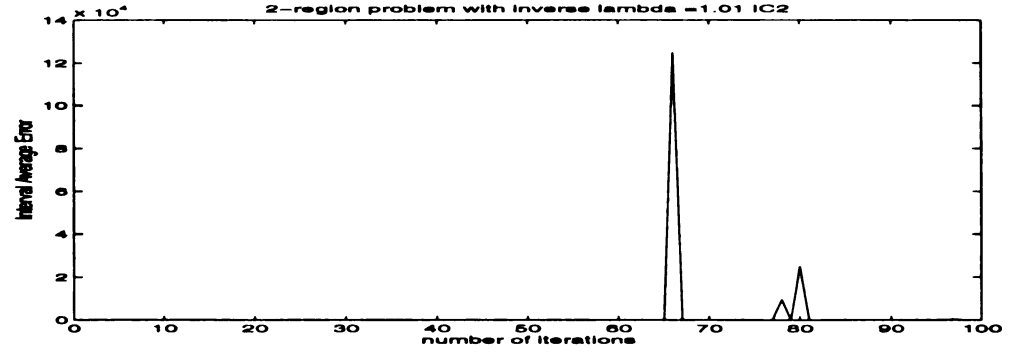


Figure 3.7. The IAE with $\lambda^{-1} = 1.01$

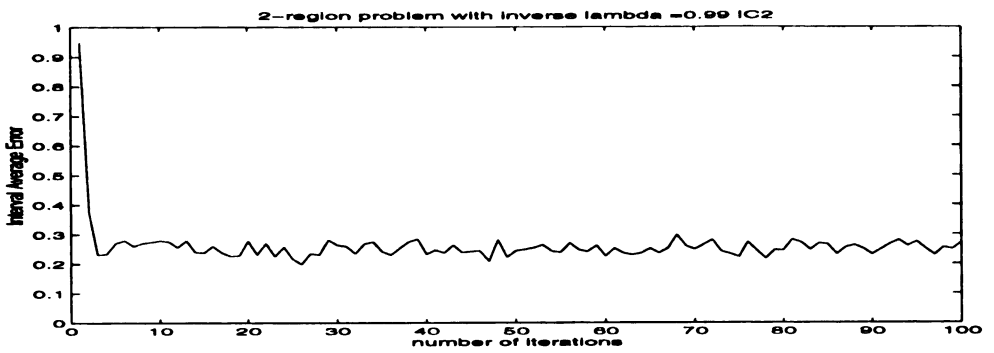


Figure 3.8. The IAE with $\lambda^{-1} = 0.99$

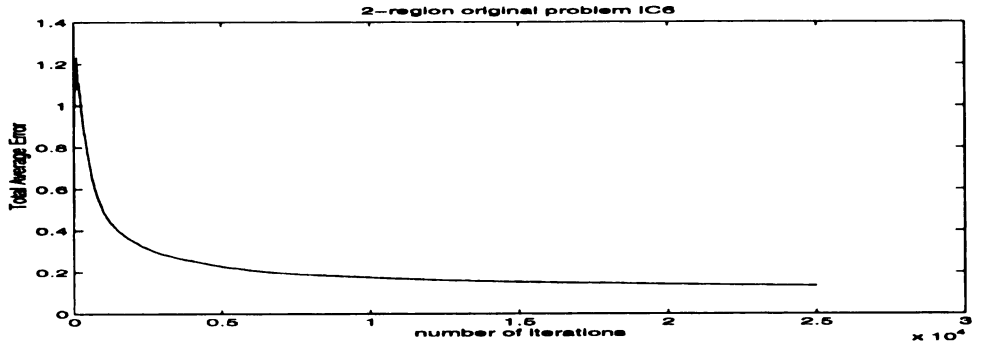


Figure 3.9. The TAE with $\lambda^{-1} = 1$

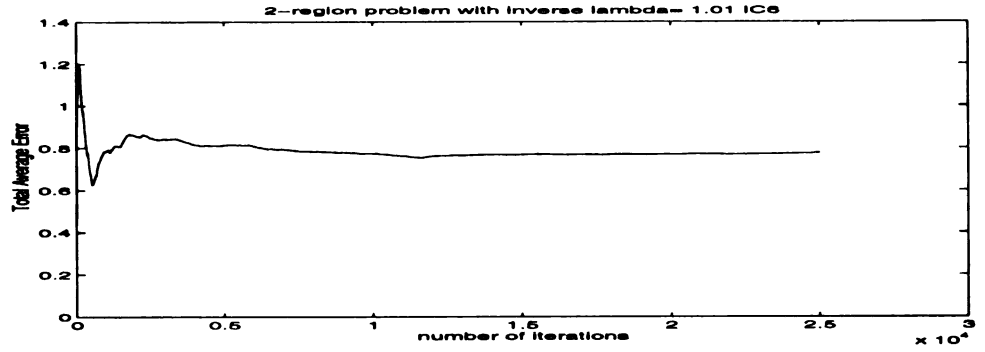


Figure 3.10. The TAE with $\lambda^{-1} = 1.01$

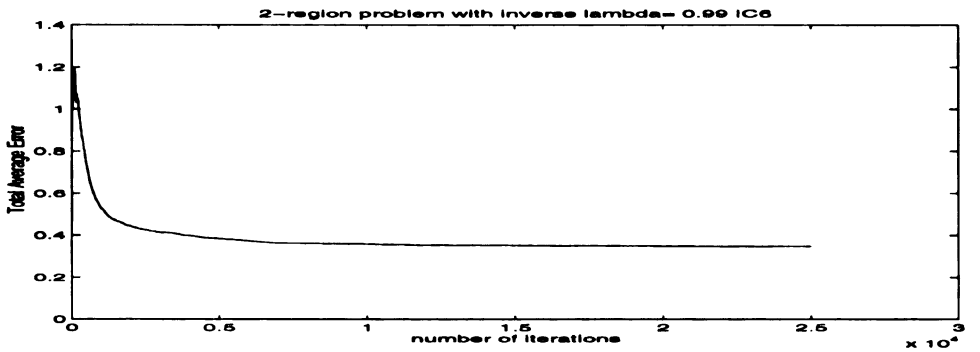


Figure 3.11. The TAE with $\lambda^{-1} = 0.99$

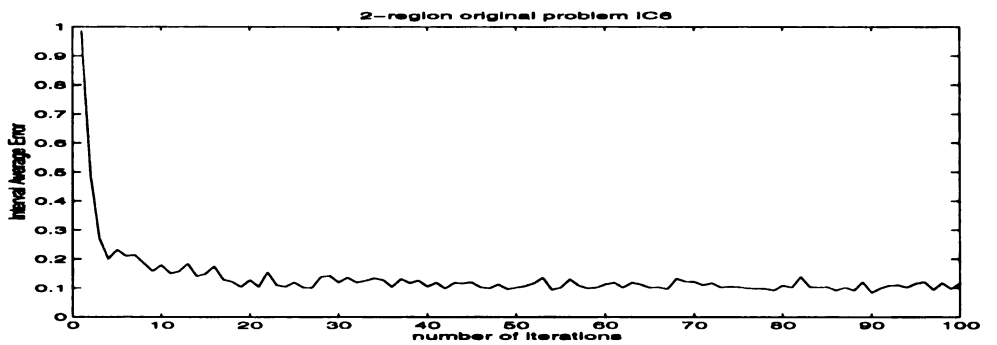


Figure 3.12. The IAE with $\lambda^{-1} = 1$

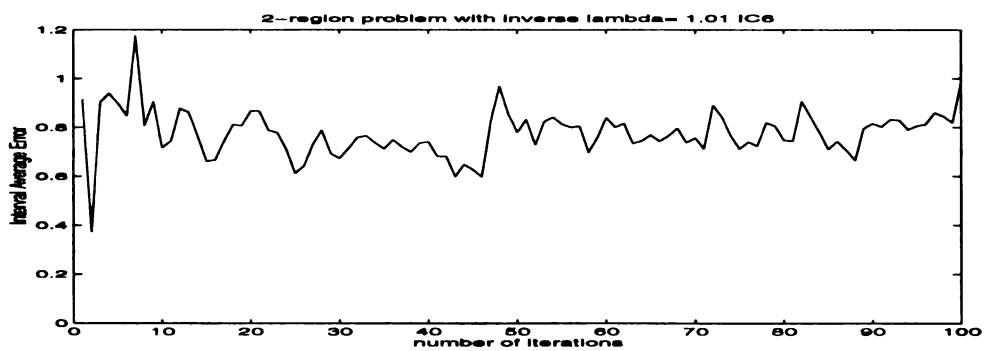


Figure 3.13. The IAE with $\lambda^{-1} = 1.01$

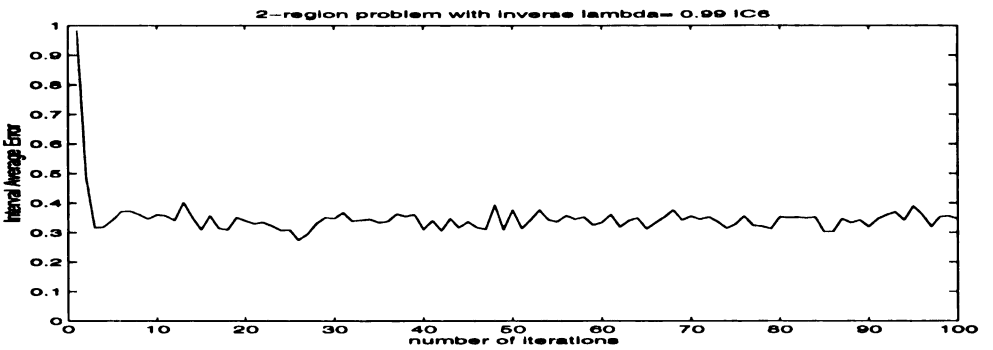


Figure 3.14. The IAE with $\lambda^{-1} = 0.99$

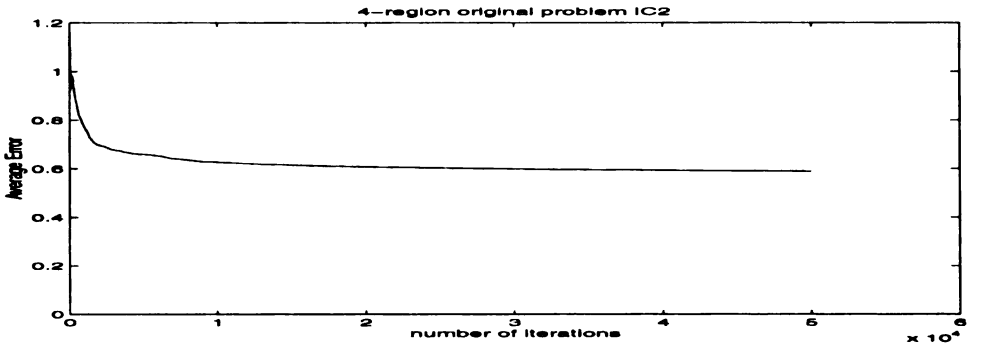


Figure 3.15. The TAE with $\lambda^{-1} = 1$

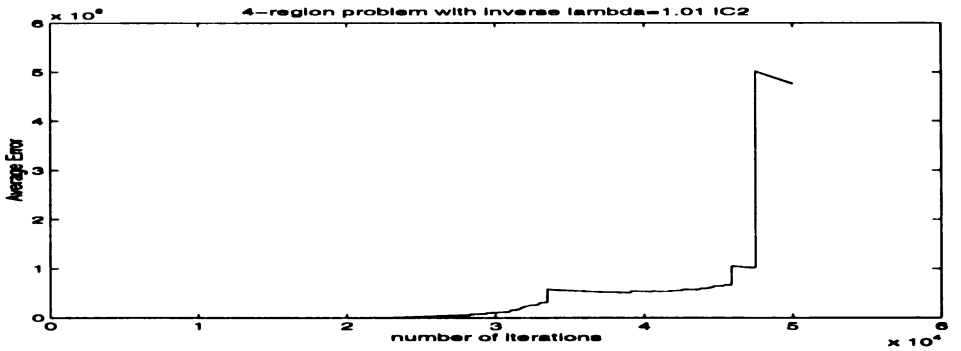


Figure 3.16. The TAE with $\lambda^{-1} = 1.01$

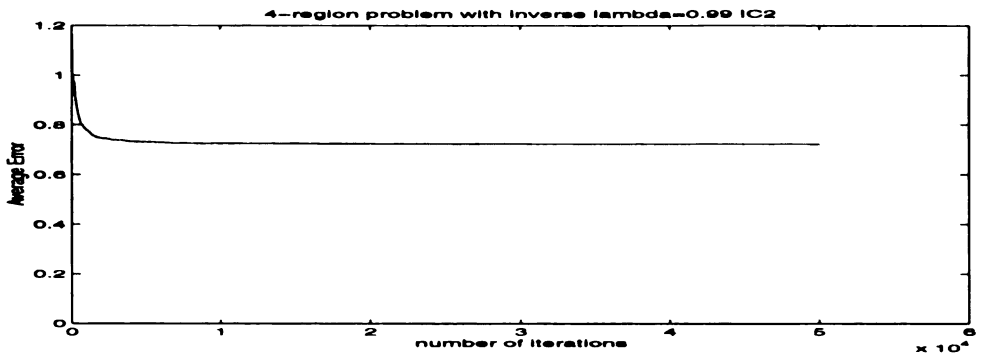


Figure 3.17. The TAE with $\lambda^{-1} = 0.99$

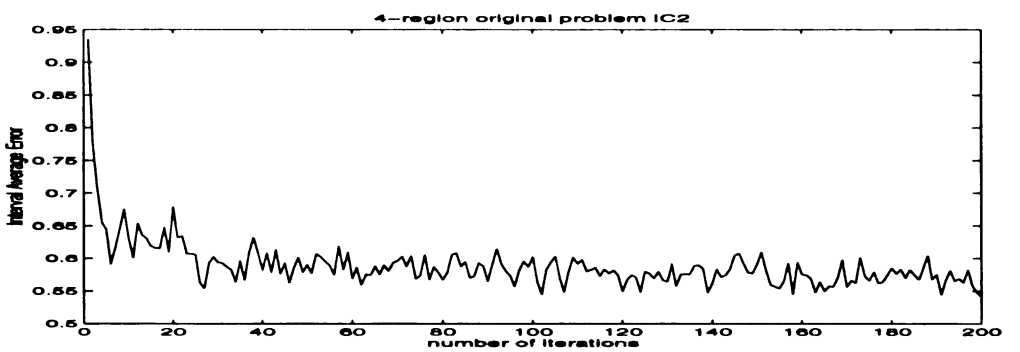


Figure 3.18. The IAE with $\lambda^{-1} = 1$

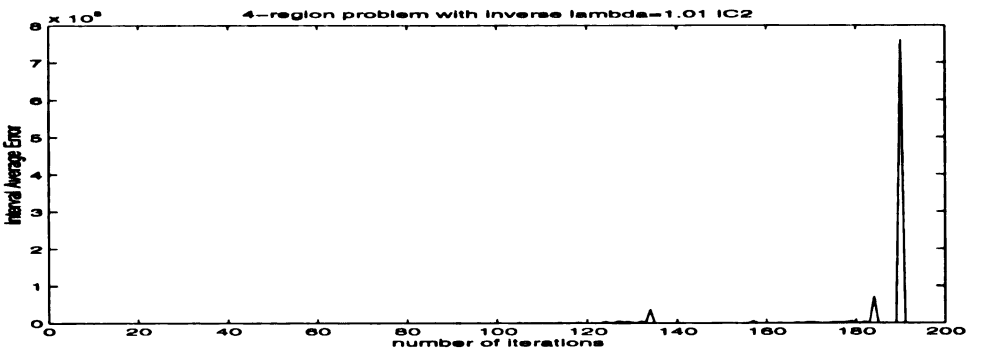


Figure 3.19. The IAE with $\lambda^{-1} = 1.01$

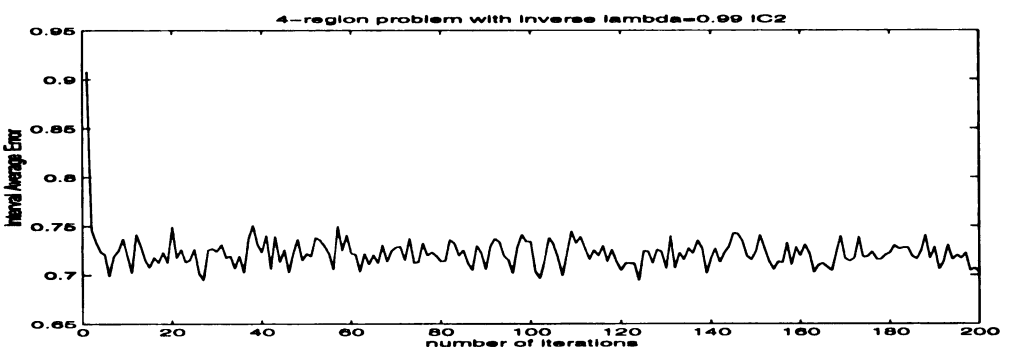


Figure 3.20. The IAE with $\lambda^{-1} = 0.99$

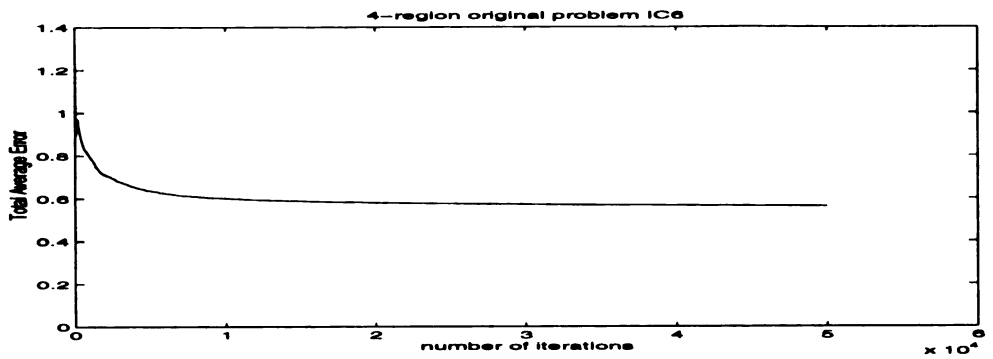


Figure 3.21. The TAE with $\lambda^{-1} = 1$

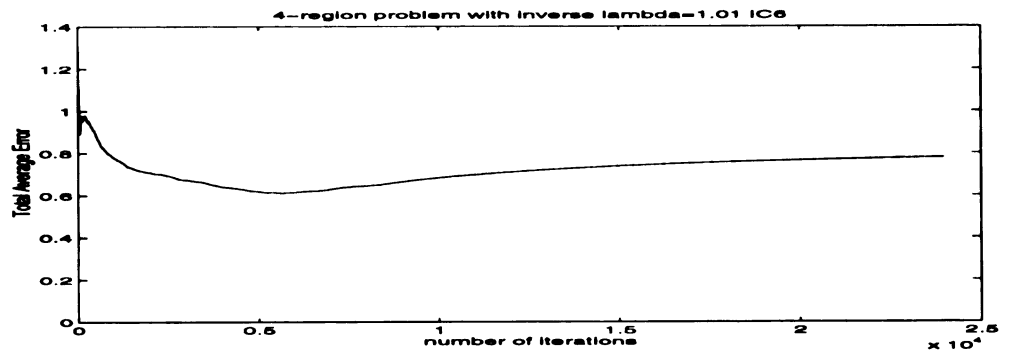


Figure 3.22. The TAE with $\lambda^{-1} = 1.01$

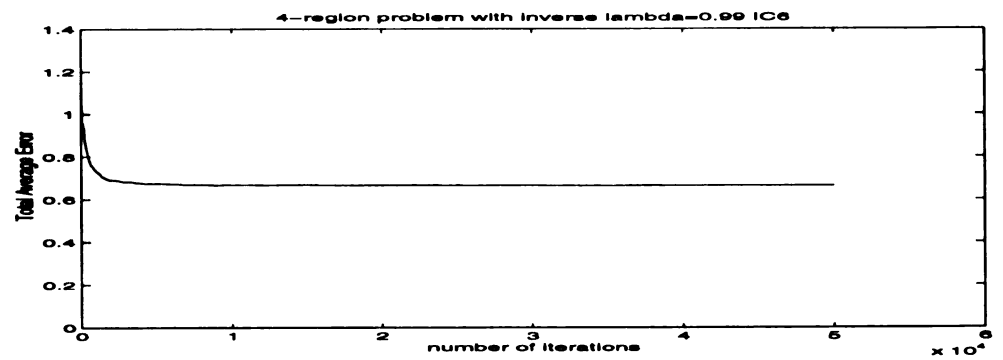


Figure 3.23. The TAE with $\lambda^{-1} = 0.99$

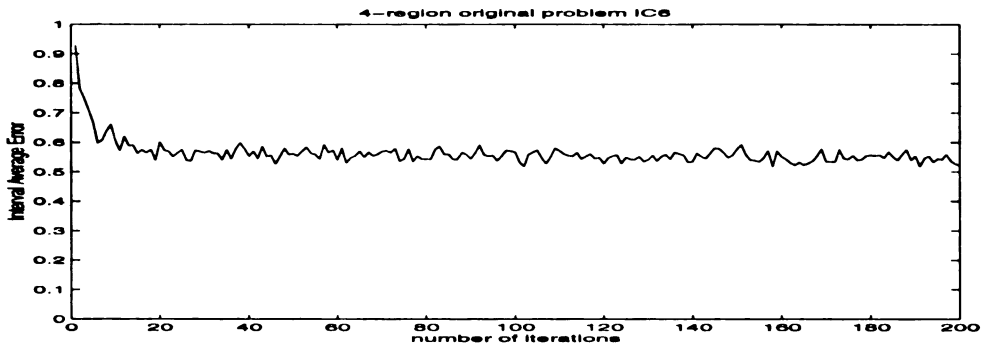


Figure 3.24. The IAE with $\lambda^{-1} = 1$

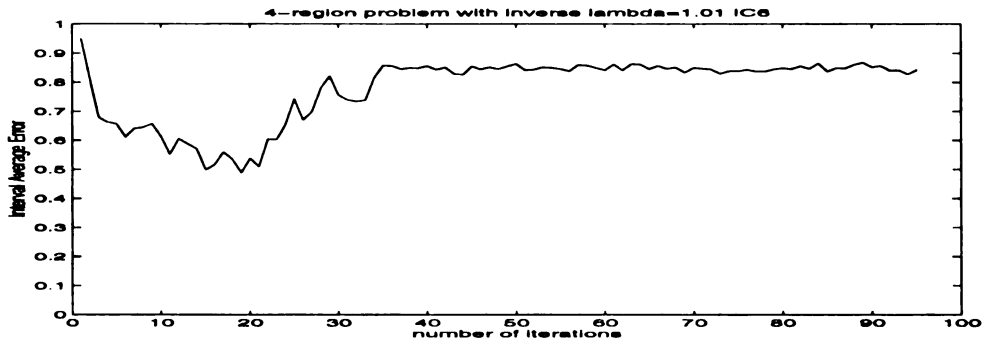


Figure 3.25. The IAE with $\lambda^{-1} = 1.01$

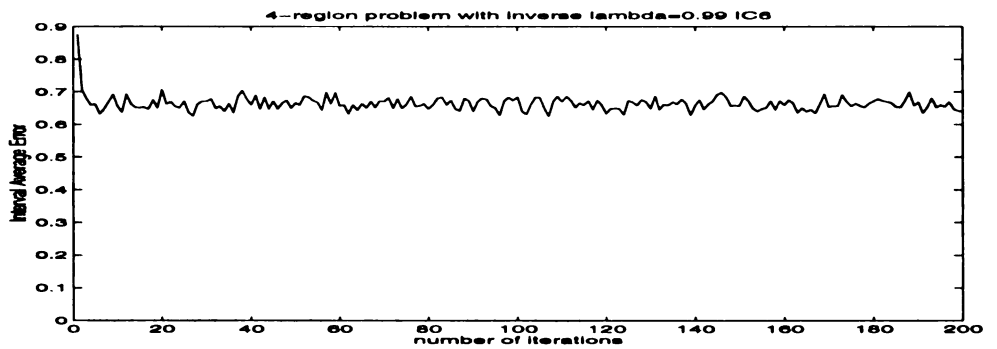


Figure 3.26. The IAE with $\lambda^{-1} = 0.99$

3.4 Discussion and Analysis of Results

The results of the simulations summarized in the above tables and figures suggest that the convergence and the stability of the Kalman filter algorithm are not very sensitive to the choice of the initial conditions. However, for different exponential weighting factors λ 's, the algorithm exhibits different behaviors. When $\lambda = 1$, the algorithm shows a stable behavior but does not converge to an acceptable local minimum. In fact the IAE as defined in the previous section, stalls for a large number of iterations. If we then choose λ slightly greater than 1, which is not in accordance with the forgetting factor definition, the algorithm is still stable but the same stalling phenomenon occurs. Note that λ grater than 1 guarantees the stability of the discrete recursion equations that include the update of the covariance matrix described in equation (3-12). λ slightly less than 1 however is a value that fits the definition of the forgetting factor and it has previously been used in other applications of the Kalman filter approach where it has performed satisfactorily. Though, in our application of the Kalman filter algorithm to supervised learning, the behavior of the algorithm under λ slightly less than 1 shows a divergence phenomenon that will be given the name "explosive divergence" in the next chapter. This behavior can be expected if we carefully observe the recursion equations of the algorithm. Equations (3-10) through (3-12) show that when $\lambda < 1$, $\lambda^{-1} > 1$ is expected to drive both the covariance matrix elements and the weight parameters to higher and higher values as the number of iterations increases.

The reason that this divergence problem occurs in this application (Supervised learning) of the algorithm and not in so many others may be due to the characteristics of the nonlinear model. The sigmoidal nonlinearity present in the (FFANN) model is the only nonlinearity present in the network. And its shape and the shape of its derivative could be at the origin of the poor performance of the algorithm. However, as described earlier in the section, divergence problems could be easily predicted from

the recursion formulas described in equations (3-10) through (3-12).

in the above conclusions the following conjecture is formulated :

3.4.1 Conjecture 1

Let $\epsilon(\hat{\theta}(n))$ be the Least Square Error defined as :

$$\epsilon(\hat{\theta}(n)) = \sum_{j=1}^{n} \| \mathbf{d}(j) - \mathbf{h}(\hat{\theta}(n), \mathbf{i}(j)) \|_2 \, \lambda^{n-j}. \tag{3.21}$$

We conjecture that the following statements are true :

- If λ is < 1 then the EKF algorithm exhibits an unstable behavior described by an explosive divergence.
- If λ is ≥ 1 then the EKF algorithm is stable but often converges to unsatisfactory local minima.

Now that the simulations performed in the previous section identified some weaknesses of the Extended Kalman filter approach to supervised learning, we propose to search for solutions for the following problem :

- How can one make it possible for the algorithm to converge to satisfactory local minima regardless of the complexity of the problem and its initial conditions. and,
- How can one improve the computational efficiency of the algorithm in cases where the performance is satisfactory but the convergence time is not adequate.

CHAPTER 4

Convergence to Satisfactory Local Minima

4.1 Motivation

The literature review done on the subject of kalman filters as an approach to supervised learning suggests that convergence to satisfactory local minima is not guaranteed and is very sensitive to the choice of initial conditions. This conclusion is drawn from the results reported in several papers [26, 25, 21]. The authors on these papers had to conduct several trials with different initial conditions then average the results. Seldom did they report all the trials with their individual results. In a practical situation, however, the averaging of the results does not give a solution to the problem. The solution is usually chosen after several trials are performed to give the best performance. The need to try different initial conditions presents an inconvenience because of the extra time and resources used to arrive to a satisfactory solution. This drawback of the algorithm has motivated the research in this chapter. The analysis of the problem begins by investigating the reasons behind the divergence of the algorithm and categorizing them. The investigation is expected to outline the steps toward a solution of the problem and is done in the second section of this chapter.

The last two sections in the chapter are designated to the proposed solutions followed by simulations on two region classification problems.

4.2 Divergence Phenomena

The kalman filter algorithm as with the special case of the recursive least squares(RLS) algorithm may exhibit two forms of divergence.

- explosive divergence
- lock-up divergence

The first form of divergence described as explosive is due to the covariance matrix $P(n)$ losing its positive definiteness which means that it becomes singular. However this kind of divergence could also occur due to roundoff errors and it is usually observed when the forgetting factor λ is less than 1. When λ is equal to 1 and the algorithm is left updating for a large number of iterations, the roundoff errors can become large and cause the algorithm to diverge.

The second form of divergence called lock-up divergence usually occurs when λ is equal to 1. This divergence shows up when the algorithm stalls. This stalling phenomena occurs when the algorithm stops updating which means that the filter weights stop changing in value. It was mentioned in [28] that The experience with this divergence may be temporary or permanent but in our own experience it was permanent. When the gain matrix which is used to update the weights becomes very small the filter weights will stop changing. However to compute the gain matrix the covariance matrix is used. If we look at the computation of the gain matrix we conclude that if the covariance matrix $P(n)$ becomes very small the gain matrix $G(n)$ also becomes small. We can then assert that the stalling phenomena is due to the covariance matrix becoming small. If the elements of the covariance matrix are near

zero the lock-up divergence is permanent.

4.3 Proposed Solutions

After having defined the two forms of divergence that are frequently present in the Extended Kalman filter algorithm, we now proceed to summarize the previous attempts from several researchers to solve such problems with divergence. We will then propose and implement our own solutions to the same problem.

The problems of divergence especially the lock-up divergence form described as the algorithm getting trapped in local minima that are often undesirable, is not only a characteristic of the extended Kalman filter algorithm. In fact the backpropagation algorithm in its original version, although guaranteed convergence, often converges to unsatisfactory local minima. Several approaches have been tried to escape these local minima. One approach was to add a momentum term to the update of the weights. Another approach which proved to be successful is the use of different energy functions or cost functions.

In the Kalman filter algorithm one technique has been used to enable escaping local minima. This technique entails the addition of an artificial process noise in the update equation of the covariance matrix. This technique was mainly used to render the covariance matrix positive definite [27]. This was the only technique that we can find in the literature that was claimed to help with the lock-up divergence. The lack of any extensive research aimed toward a solution to the two forms of divergence described in the previous section is what motivated our research in this chapter. We will now begin outlining our own strategy for finding a solution to this problem.

The Kalman filter algorithm as applied to supervised learning has been derived by considering a specific energy function for which the goal is to find a set of optimal parameters that will minimize it. The energy function is usually defined as a mean

square error between a desired output or target and the actual output given by the network. This (LSE) could also be weighted by a suitable function or coefficient [29]. This weight factor has proven to be beneficial in terms of improving the convergence. In our work we investigate the influence of the weighting factor λ on the convergence of the algorithm. Even though λ was always present in the recursion formulas of the algorithm in the earlier work [26] it was kept fixed at λ equal to 1. Based on the theoretical analysis of the stability and the convergence problems the algorithm can exhibit [8], it would be safe to keep λ fixed at 1. However this is a very conservative choice and consequently the algorithm can present a stalling phenomena that is usually observed around an unsatisfactory local minimum.

We concentrate on the deterministic formulation, for which the estimates are obtained from the minimization of

$$\epsilon(n) = \sum_{j=1}^{n} \| \mathbf{e}(j) \|_2^2 \, \lambda^{n-j}. \tag{4.1}$$

We propose an adaptation scheme for λ which depends on how fast the error is decreasing and whether a stalling phenomena is present or not. We keep λ smaller or equal to 1 as necessary. Preliminary results on two classical classification problems are very promising. Varying λ has enabled the network to automatically escape from the local minimum and decrease the error to a better minimum. A comparison of the algorithm with or without fixing λ using the same set of initial conditions will be given.

4.3.1 A Modified update procedure

The original (NDEKF) used a forgetting factor that was fixed at 1. This value of λ guarantees stability without always guaranteeing a good performance. To solve this problem we propose the following modified updates based on varying λ. After the

mean squared error is averaged every 250 iterations (We call this average: an interval average error), this error is then compared to the previous error. If the difference is smaller than a certain threshold, it is decided that the network is stalling. At that moment λ is changed to a value that is smaller but still very close to 1. If this new value is kept unchanged for a large number of iterations, the network shows some instability behavior. This phenomenon is justifiable if we examine the update formulas. In order to guarantee the stability a mechanism that will switch λ back to 1 had to be incorporated in the algorithm. Three different ways were considered :

1. One choice is to change λ for only a few iterations.
2. A second choice is to change λ until the total average error shows some increase over the previous average error.
3. A third choice is to change λ until the instantaneous error increases as compared to the one computed 1000 iterations before.

4.3.2 The forgetting factor as a function of error

Since the mechanism of switching λ back and forth between 1 and less than 1 depends on how the so called IAE behaves (whether it is decreasing or stalling), it would be convenient to model λ as a function of this IAE. The inconvenience of the other choices is manifested in one of them at least. The time to switch λ back to 1 was done by trial and error and that could take few trials. However from the few experimental trials we could conclude that as long as we only switch λ for less than 50 iterations the stability is preserved. A suitable function that would model the behavior of λ as a function of the error has to satisfy at least these two conditions in order to guarantee stability.

- λ is slightly < 1 when the difference between two consecutive interval average errors is less than a certain threshold.

- λ is 1 when the difference defined above exceeds the same threshold chosen above.

Several functions were picked but the one that was closest in satisfying the conditions is the following:

$$\lambda = \begin{cases} \tanh(\alpha * \delta) & \text{if } \delta > \gamma \\ 0.98 & \text{otherwise} \end{cases} \tag{4.2}$$

where δ is defined

$$\delta = (\sum_{j=k+1}^{k+1+L} \| \mathbf{e}(j) \|_2)/L - (\sum_{j=k}^{k+L} \| \mathbf{e}(j) \|_2)/L. \tag{4.3}$$

γ is a difference of error threshold chosen as 0.0025 in our simulations. It could be changed accordingly with the problem. and $k = i/L$ and i is the number of iterations that is a multiple of L. In our simulations L was chosen as the value 250 for comparison purposes. α is decided accordingly with the desired behavior of λ. In our simulations $\alpha = 1000$ fits best with the desired conditions for the chosen function.

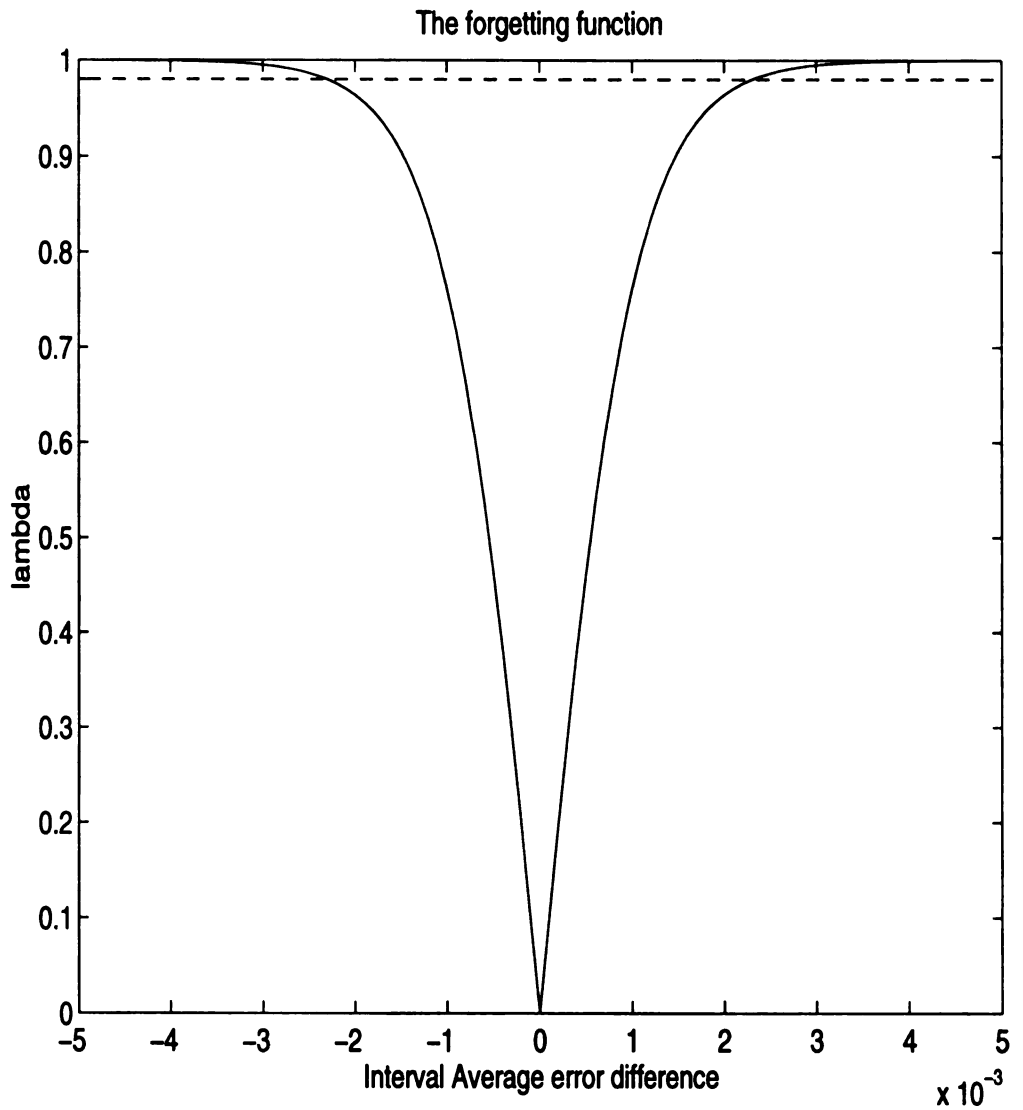


Figure 4.1. $\lambda = \tanh(\alpha * \delta)$ with $\alpha = 1000$ and δ is the IAE difference

Table 4.1. The 2-region classification problem

update choices		IC1	IC2	IC3	IC4	IC5	IC6
$\lambda = 1$	TAE	0.1095	0.1255	0.1102	0.1326	0.1136	0.1348
	IAE	0.0823	0.1069	0.1022	0.1061	0.1006	0.1176
	class %	97.2	96.2	97.0	96.2	97.0	95.8
	#iterations	25000	25000	25000	25000	25000	25000
choice1	TAE	0.1550	0.1452	0.1275	0.1240	0.1136	0.1447
	IAE	0.0409	0.0413	0.0468	0.0493	0.1006	0.0500
	class %	96.6	98.2	98	98	97	96.9
	#iterations	6500	9750	10250	19500	25000	19000
choice2	TAE	0.2052	0.2419	0.1864	0.2157	0.2047	0.2502
	IAE	0.0427	0.1236	0.0329	0.0987	0.0756	0.0839
	class %	97.6	95.6	97.5	96.0	96.4	96.2
	#iterations	4750	3500	4750	6500	4750	3500
choice3	TAE	0.1165	0.12	0.1128	0.1291	0.1319	0.1399
	IAE	0.0374	0.0687	0.0459	0.0780	0.0693	0.0818
	class %	97.0	98.0	97.2	97.1	96.4	96.4
	#iterations	19750	25000	22000	25000	14500	21000
choice4	TAE	0.1250	0.2667	0.1179	0.1503	0.1998	0.1602
	IAE	0.0450	0.1388	0.1084	0.0661	0.0660	0.0573
	class %	98.6	95.6	97.5	96.7	96.8	97.2
	#iterations	12000	25000	25000	13750	4750	11750

Table 4.2. The 4-region classification problem

update choices		IC1	IC2	IC3	IC4	IC5	IC6
$\lambda = 1$	TAE	0.5757	0.5885	0.5718	0.5357	0.5596	0.5624
	IAE	0.5736	0.5416	0.5295	0.4916	0.5145	0.5210
	class %	76.9	75.1	77.7	77.1	78.2	78.4
	#iterations	50000	50000	50000	50000	50000	50000
choice1	TAE	0.4355	0.4741	0.4356	0.4537	0.4917	0.4983
	IAE	0.2892	0.29	0.2811	0.27	0.2822	0.2851
	class %	93.1	93.6	91.7	95.3	95.5	94.8
	#iterations	22250	50000	23000	41750	29750	30500
choice2	TAE	0.5229	0.5197	0.3027	0.4869	0.4413	0.4851
	IAE	0.2966	0.2973	0.2673	0.2899	0.2893	0.2976
	class %	91.9	92.6	93.9	93.1	95.6	92.7
	#iterations	16250	20750	26750	25250	26750	24000
choice3	TAE	0.4370	0.5146	0.4632	0.4555	0.4700	0.5137
	IAE	0.3483	0.4260	0.3648	0.3566	0.3868	0.4282
	class %	93.5	87.1	87.5	90.1	90.0	89.9
	#iterations	50000	50000	50000	50000	50000	50000
choice4	TAE	0.5288	0.5132	0.4696	0.4862	0.4676	0.4534
	IAE	0.2891	0.3430	0.3859	0.2639	0.2764	0.2794
	class %	93.9	92.1	87.3	92.1	93.06	94.0
	#iterations	23750	22500	50000	26750	50000	37250

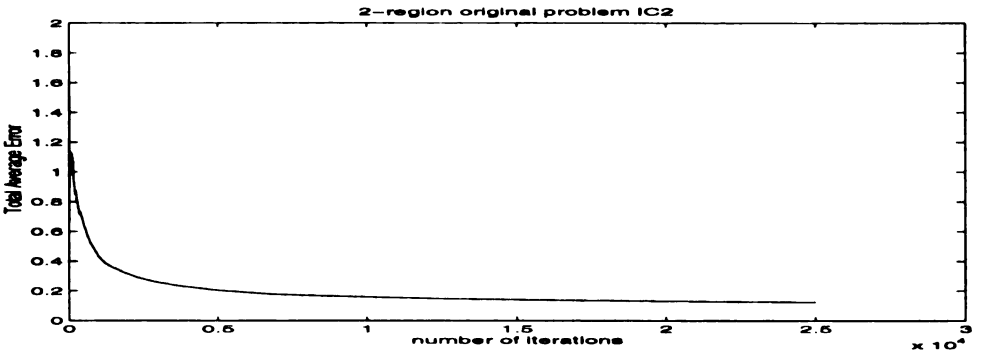


Figure 4.2. The TAE for $\lambda^{-1} = 1$

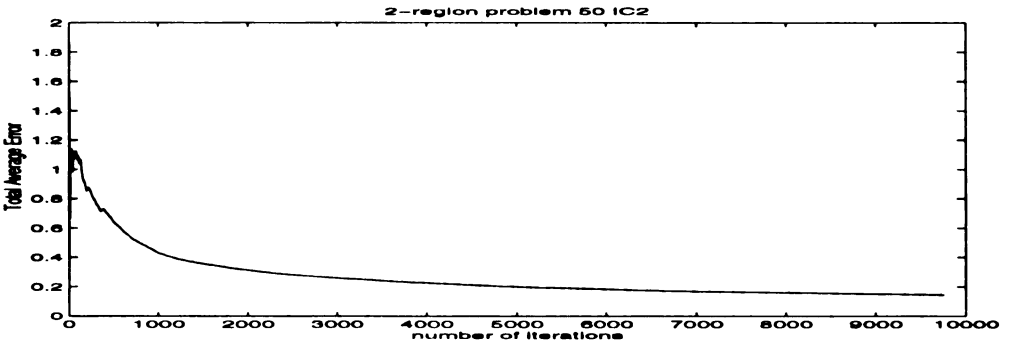


Figure 4.3. The TAE for update choice1

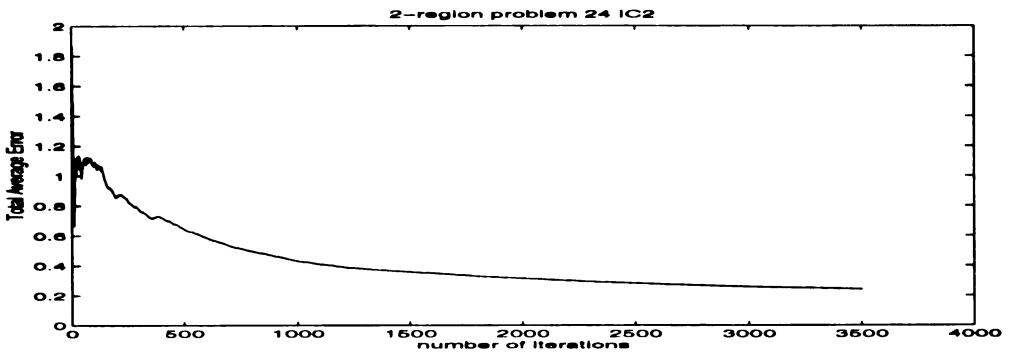


Figure 4.4. The TAE for update choice2

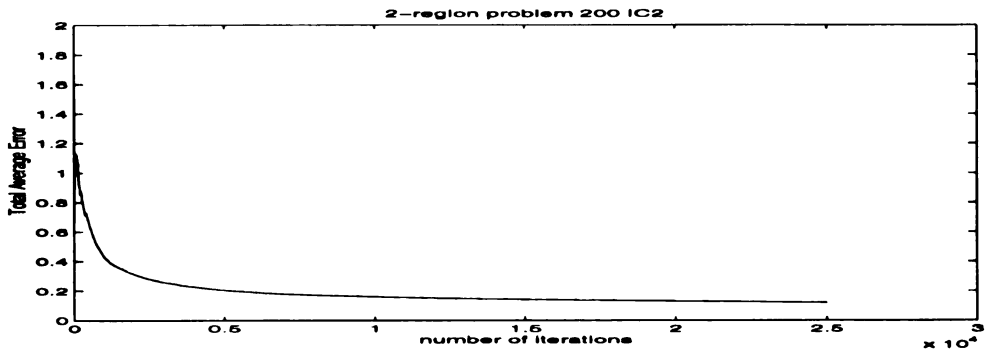


Figure 4.5. The TAE for update choice3

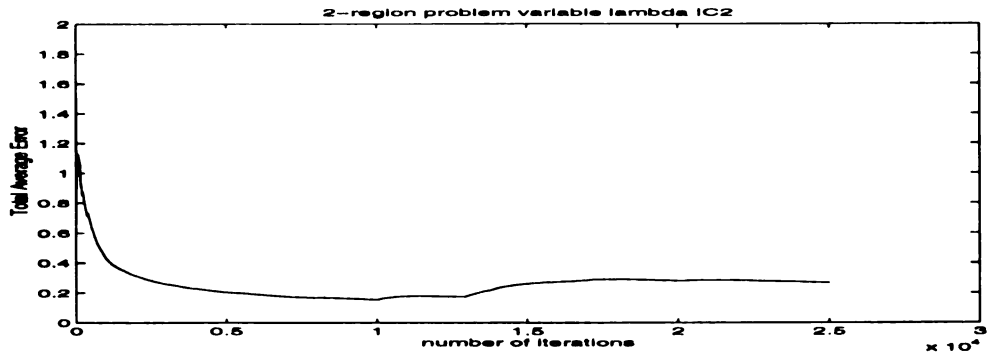


Figure 4.6. The TAE for update choice4

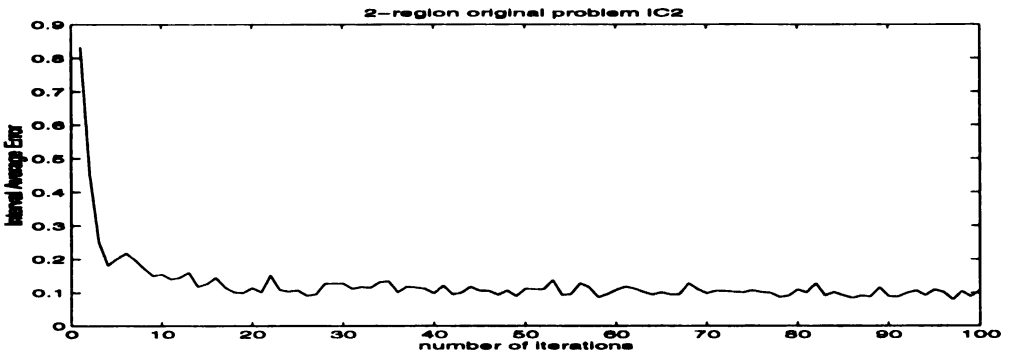


Figure 4.7. The IAE for $\lambda^{-1} = 1$

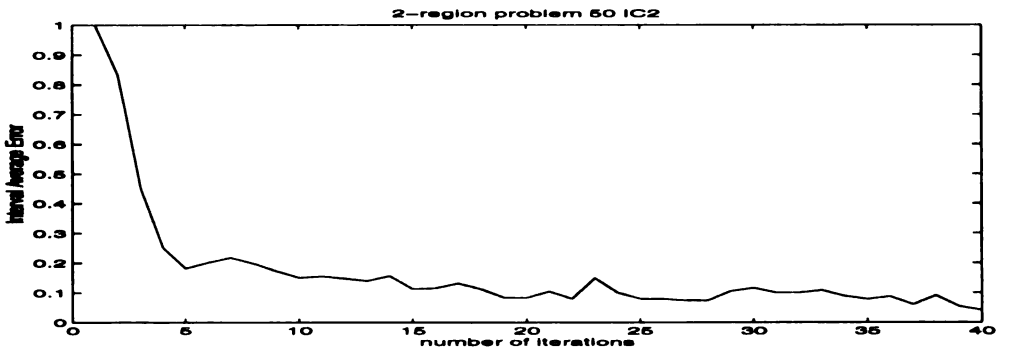


Figure 4.8. The IAE for update choice1

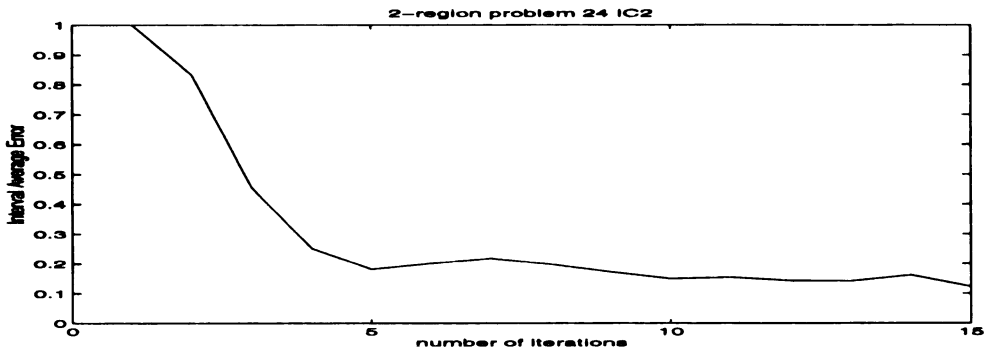


Figure 4.9. The IAE for update choice2

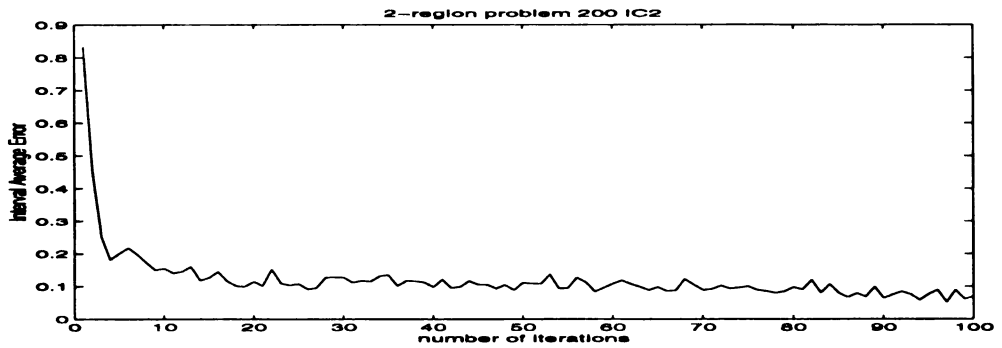


Figure 4.10. The IAE for update choice3

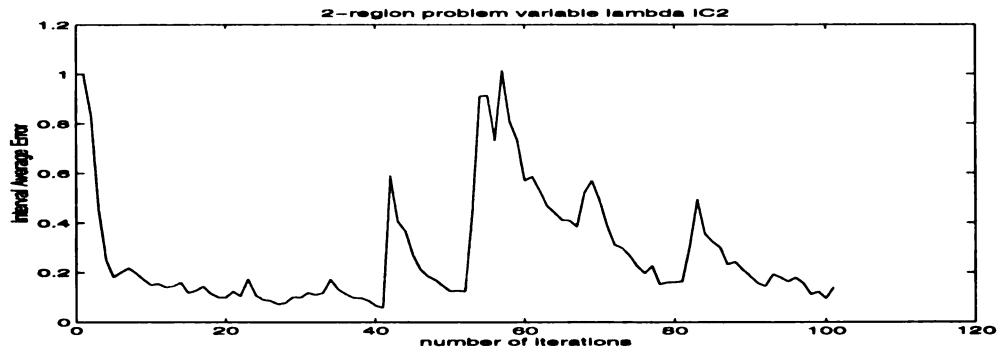


Figure 4.11. The IAE for update choice4

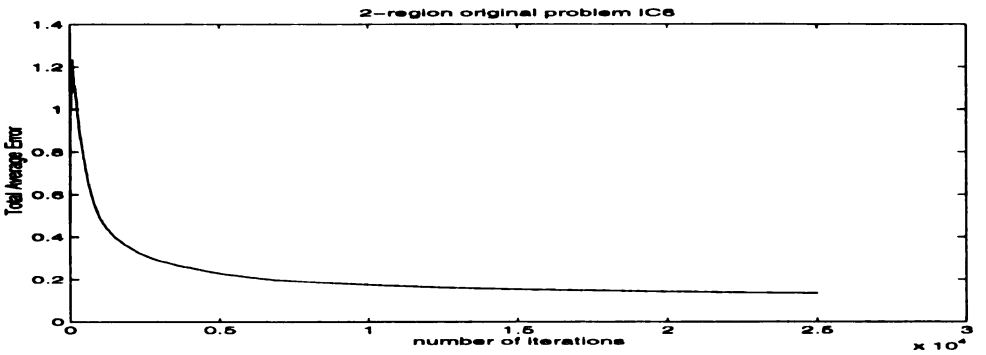


Figure 4.12. The TAE for $\lambda^{-1} = 1$

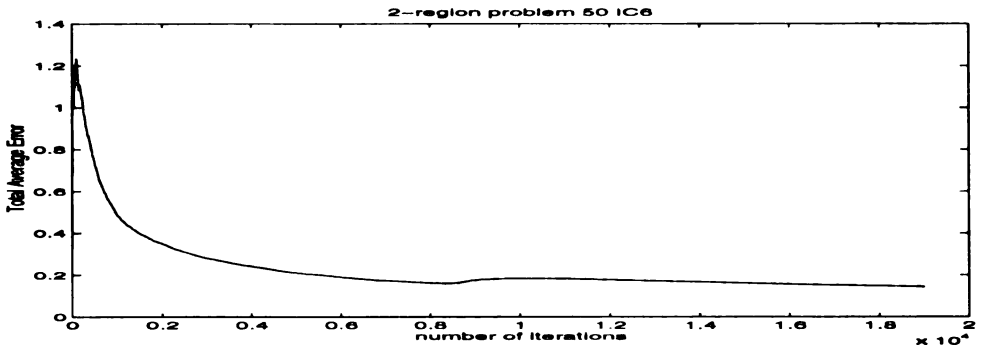


Figure 4.13. The TAE for update choice1

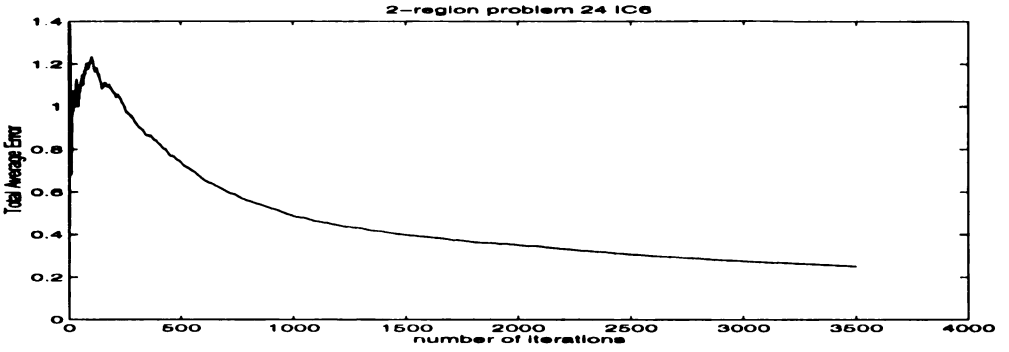


Figure 4.14. The TAE for update choice2

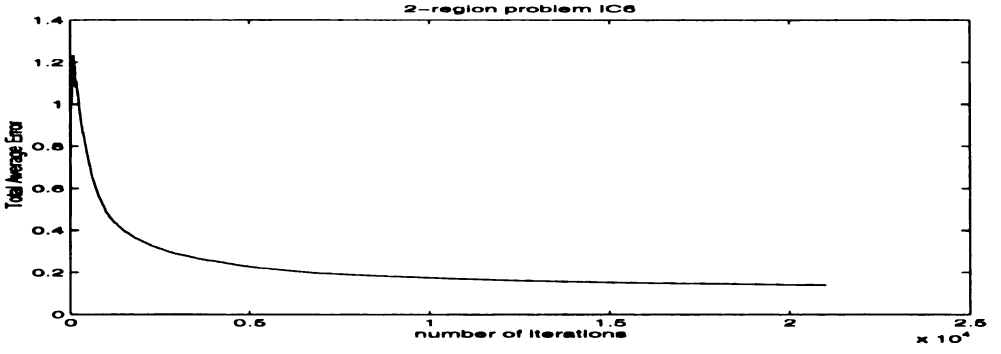


Figure 4.15. The TAE for update choice3

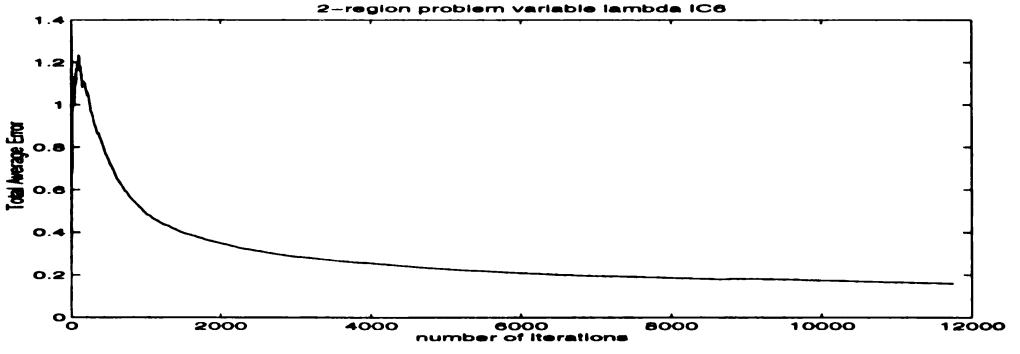


Figure 4.16. The TAE for update choice4

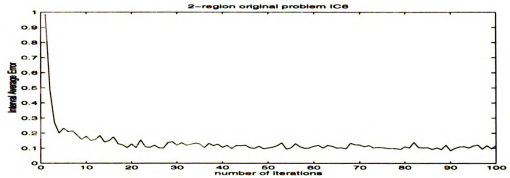


Figure 4.17. The IAE for $\lambda^{-1} = 1$

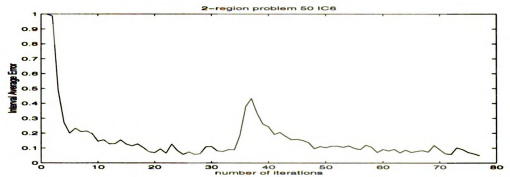


Figure 4.18. The IAE for update choice1

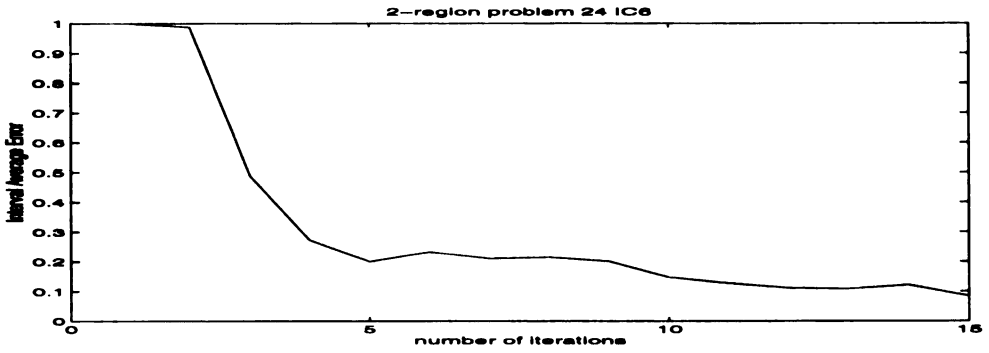


Figure 4.19. The IAE for update choice2

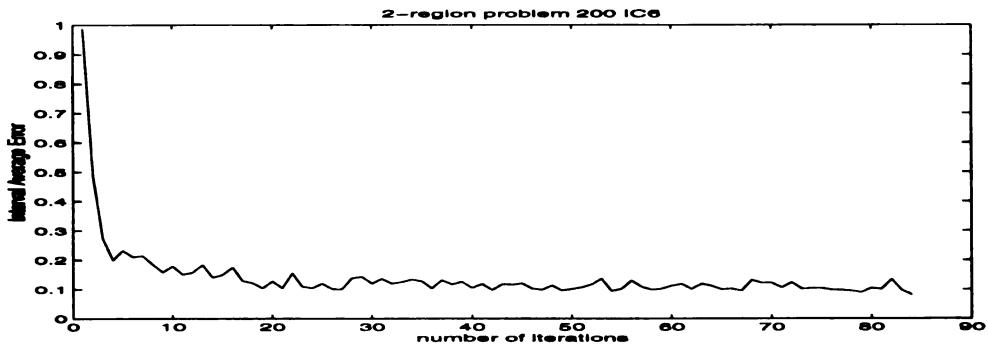


Figure 4.20. The IAE for update choice3

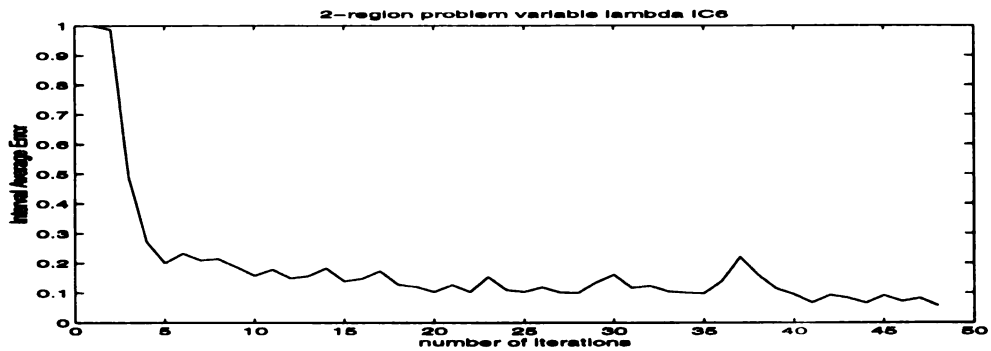


Figure 4.21. The IAE for update choice4

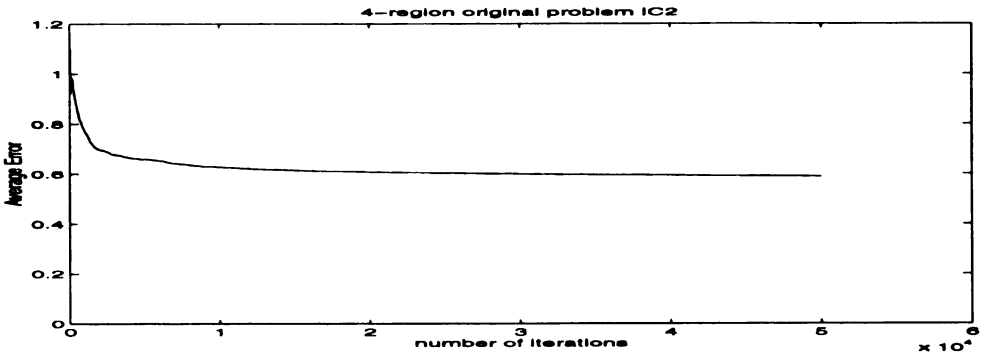


Figure 4.22. The TAE for $\lambda^{-1} = 1$

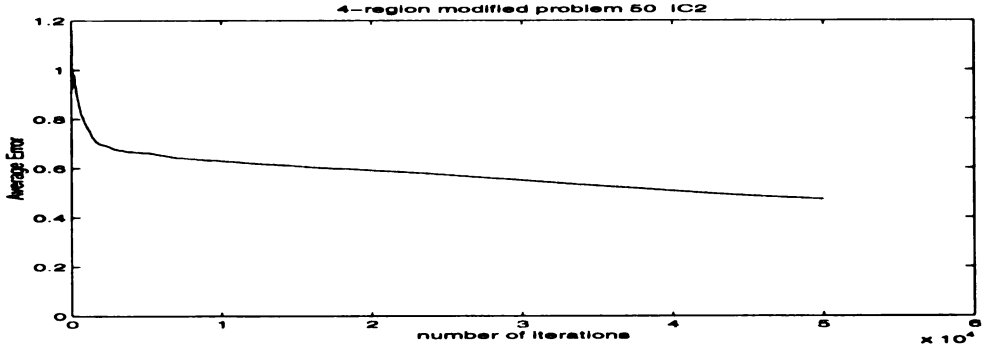


Figure 4.23. The TAE for update choice1

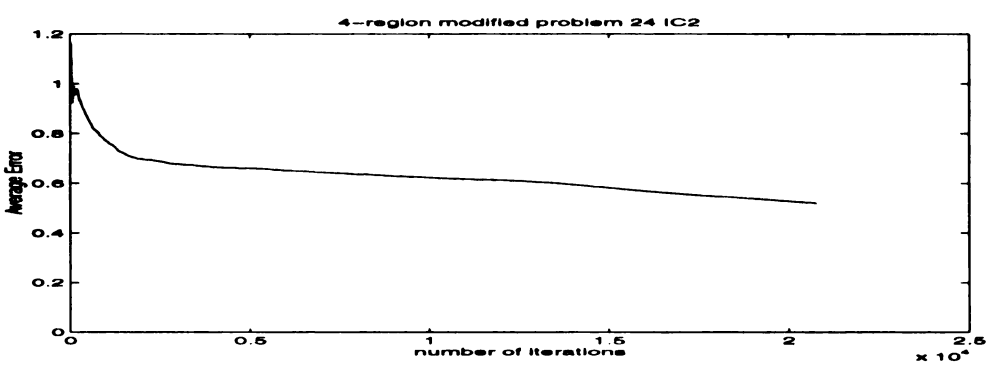


Figure 4.24. The TAE for update choice2

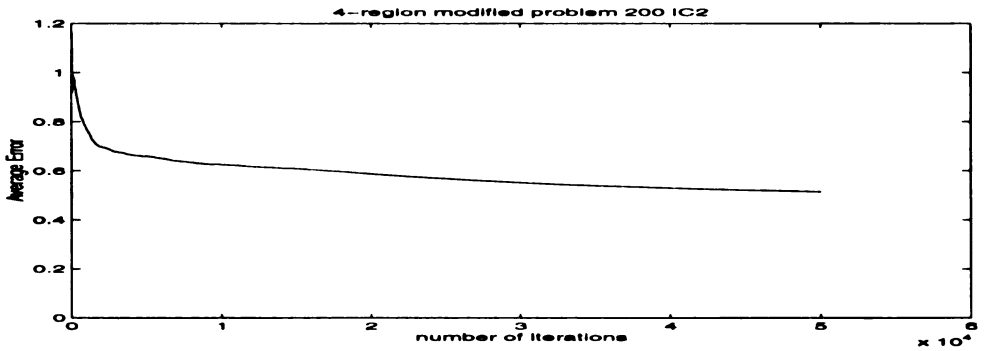


Figure 4.25. The TAE for update choice3

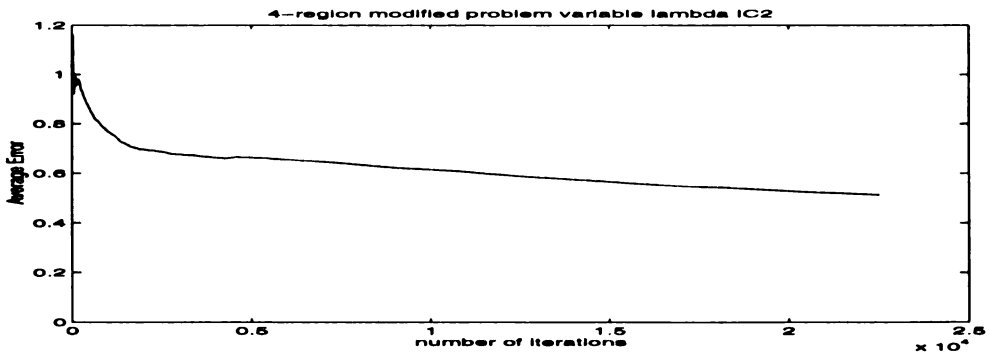


Figure 4.26. The TAE for update choice4

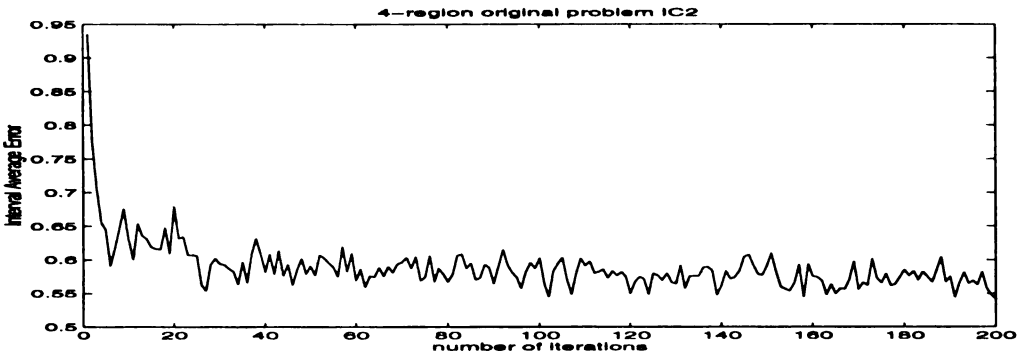


Figure 4.27. The IAE for $\lambda^{-1} = 1$

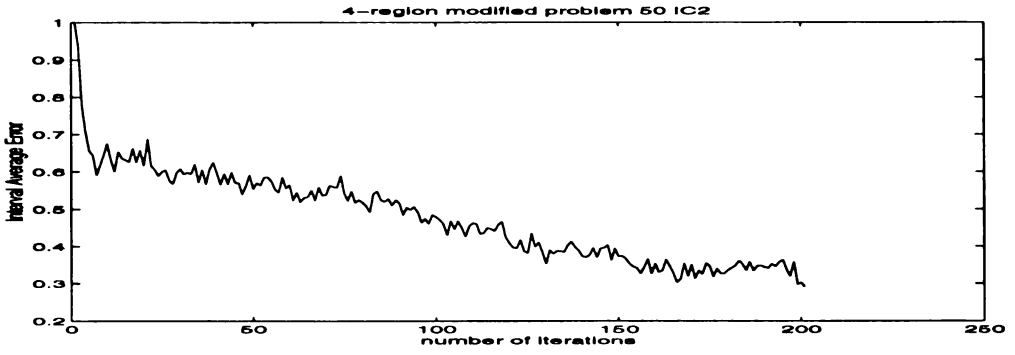


Figure 4.28. The IAE for update choice1

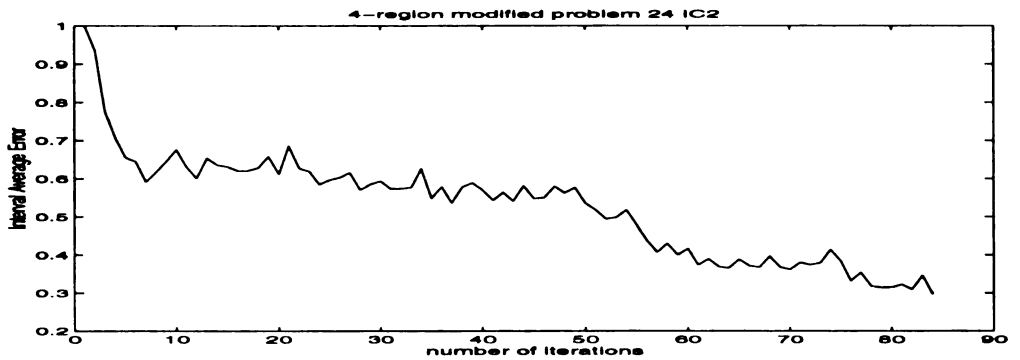


Figure 4.29. The IAE for update choice2

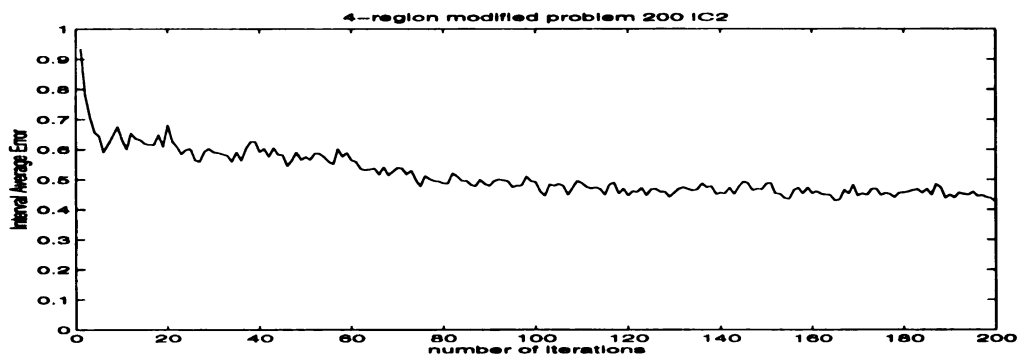


Figure 4.30. The IAE for update choice3

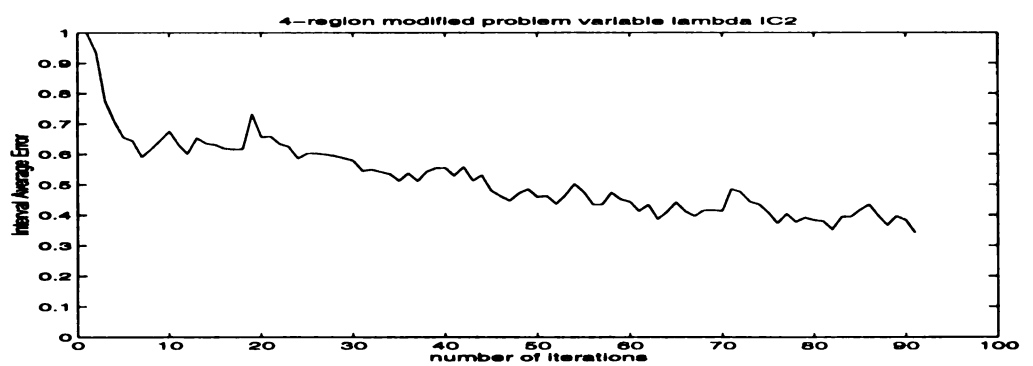


Figure 4.31. The IAE for update choice4

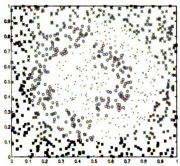


Figure 4.32. The 4-region classification pattern

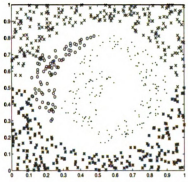


Figure 4.33. Classification Performance for $\lambda^{-1} = 1$

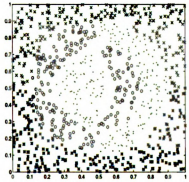


Figure 4.34. Classification Performance for update choice1

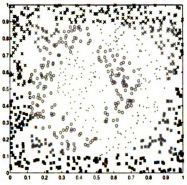


Figure 4.35. Classification Performance for update choice2

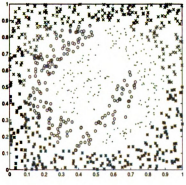


Figure 4.36. Classification Performance for update choice3

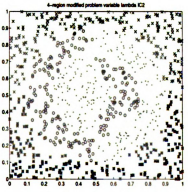


Figure 4.37. Classification Performance for update choice4

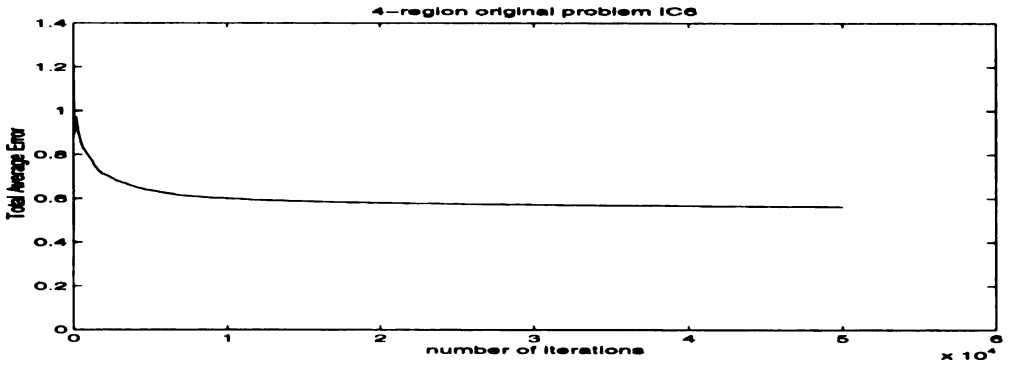


Figure 4.38. The TAE for $\lambda^{-1} = 1$

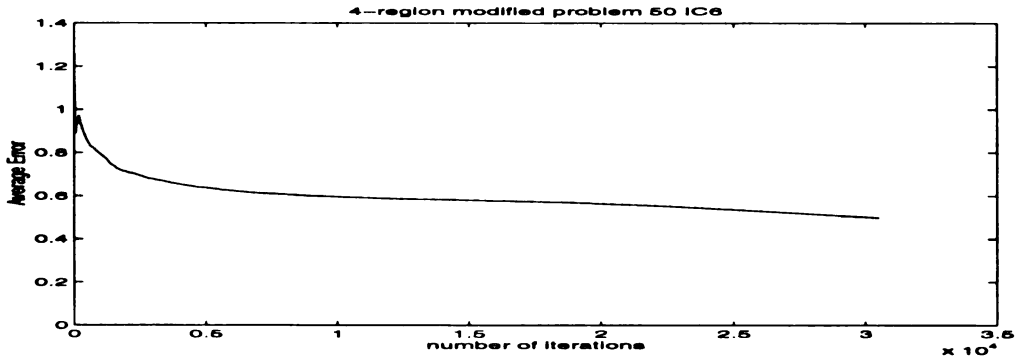


Figure 4.39. The TAE for update choice1

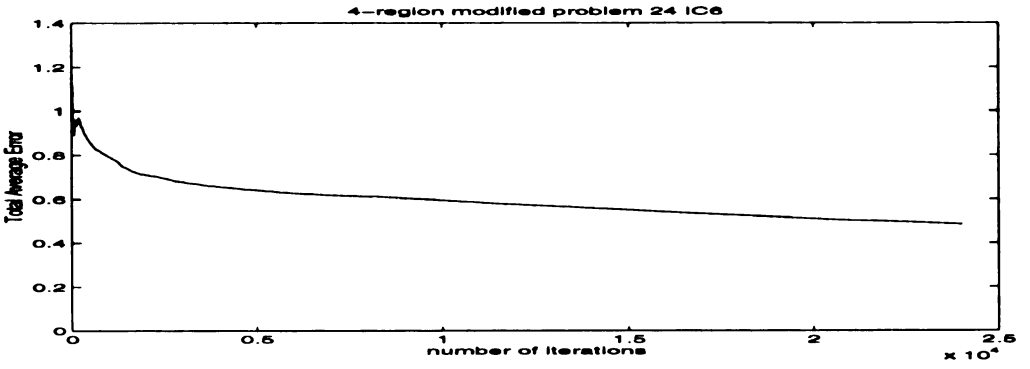


Figure 4.40. The TAE for update choice2

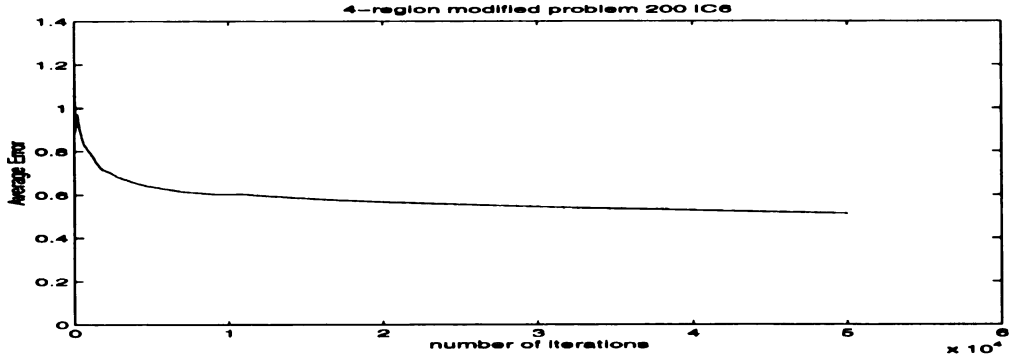


Figure 4.41. The TAE for update choice3

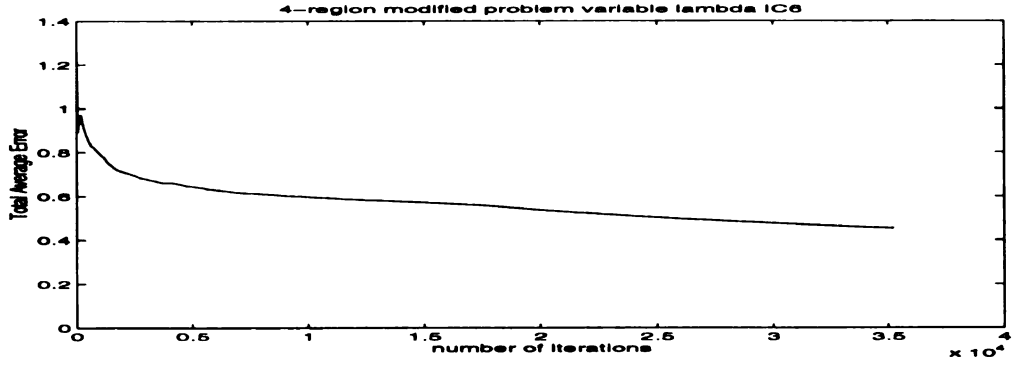


Figure 4.42. The TAE for update choice4

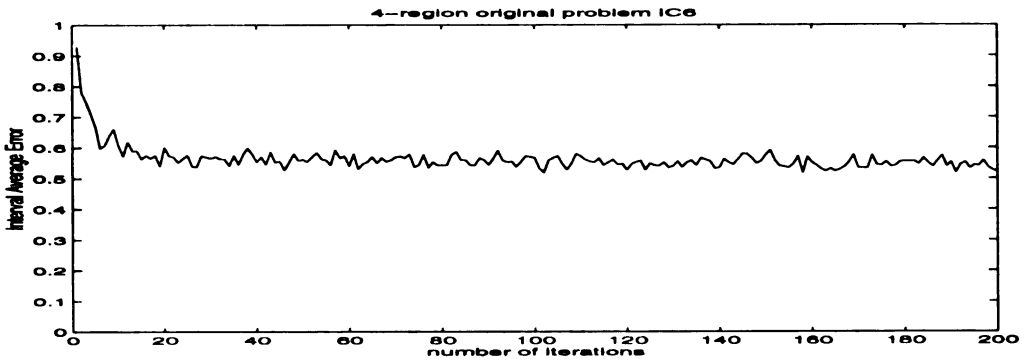


Figure 4.43. The IAE for $\lambda^{-1} = 1$

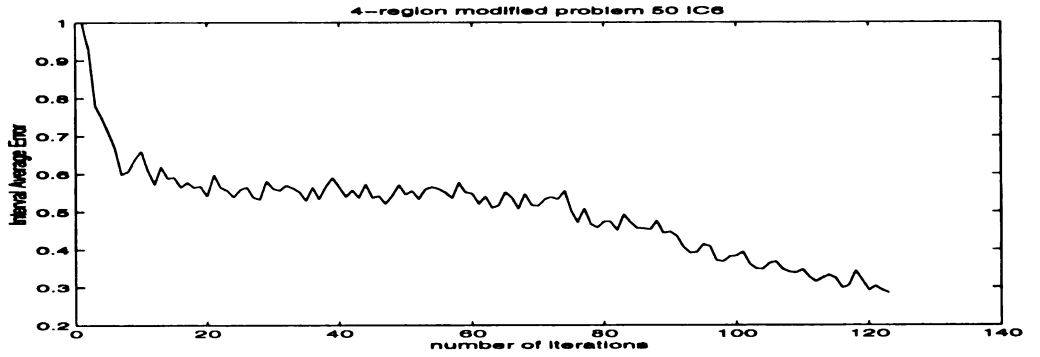


Figure 4.44. The IAE for update choice1

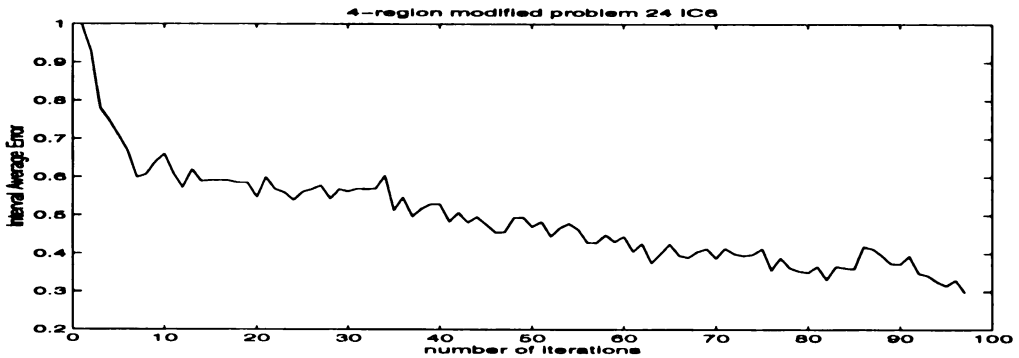


Figure 4.45. The IAE for update choice2

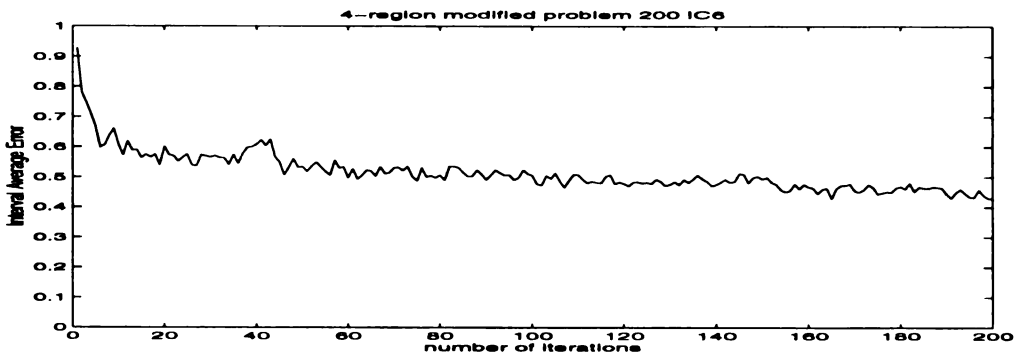


Figure 4.46. The IAE for update choice3

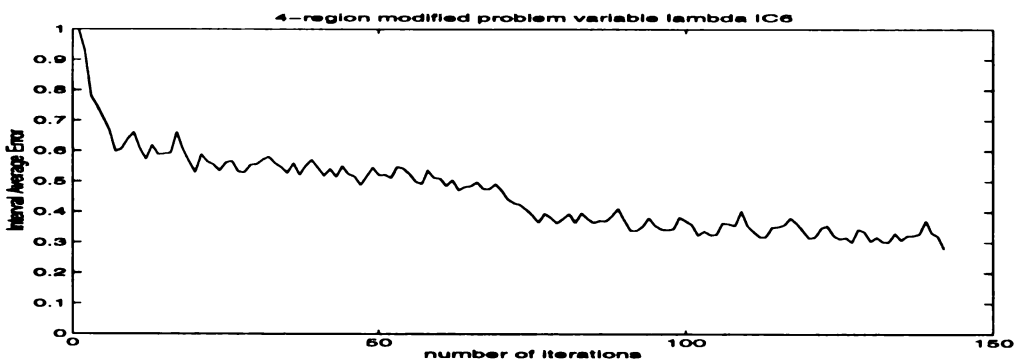


Figure 4.47. The IAE for update choice4

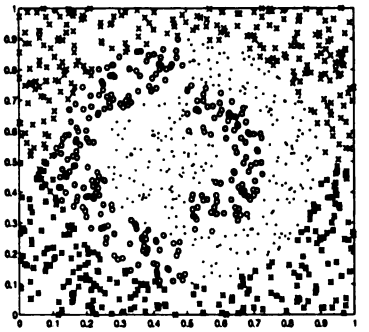


Figure 4.48. The 4-region classification pattern

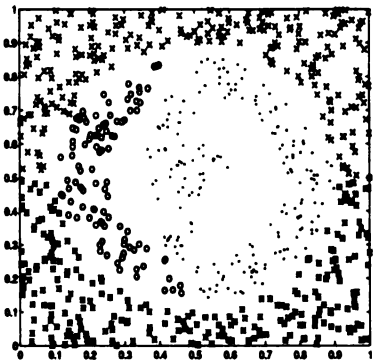


Figure 4.49. Classification Performance for $\lambda^{-1} = 1$

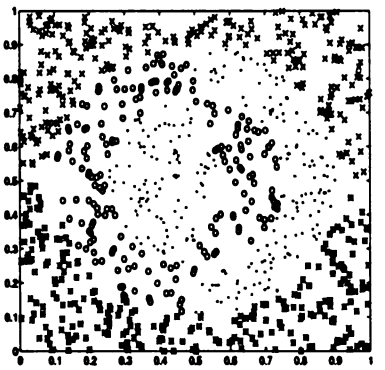


Figure 4.50. Classification Performance for update choice1

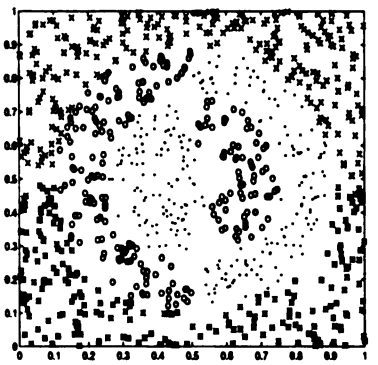


Figure 4.51. Classification Performance for update choice2

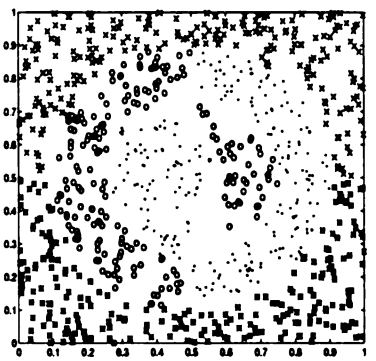


Figure 4.52. Classification Performance for update choice3

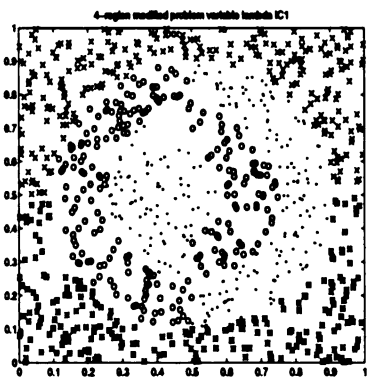


Figure 4.53. Classification Performance for update choice4

4.4 Discussion and Categorization of Results

The results of the simulations summarized in the above tables and figures suggest that using a fixed forgetting factor λ is the cause of both types of divergences. The case of $\lambda = 1$ shows a lock-up divergence problem while the case of $\lambda < 1$ exhibits an explosive divergence phenomenon. Let's now analyze the results of the different update choices closely. In the case of the 2-region classification problem, most of the update choices suggested decreased the number of iterations needed for an acceptable local minimum without degrading the performance. In this case, what is gained is the computational efficiency of the algorithm. Choice4 of the updates in this case did not work as well as the others but can be improved if the parameters of the suggested forgetting function are adjusted accordingly with the initial conditions. In most methods that are suggested for improving any type of algorithm, there is always tunable parameters that need to be dealt with.

The more complex case of the 4-region classification problem is where the proposed update choices prove to considerably improve the algorithm. In fact, most of the update choices escape from unsatisfactory local minima to search for more satisfactory ones. When a stalling phenomena is observed for an interval of some number of iterations, the algorithm reacts automatically and decides to update λ accordingly. Convergence to satisfactory local minima is then achieved at a faster rate which not only improves the performance of the algorithm but also its computational efficiency.

Choice4 of the updates gives the algorithm more flexibility by allowing the user to decide what parameters of the forgetting function fit best with his/her problem. One is able to make the decision after only a single simulation trial on his/her problem.

In summary, these different choices of updates show a satisfactory performance even though some are better than others. The advantage of having a variety of updates to pick from is an increase of the chance of success in achieving satisfactory

local minima for any given initial condition. Even if choice1 through choice3 don't achieve the goal, choice4 can be made to converge by a simple adjustment of the parameters of the forgetting function. (i.e α,L and γ)

Based in the above conclusions the following conjecture is formulated :

4.4.1 Conjecture 2

Let $\epsilon(\hat{\theta}(n))$ be the Least Square Error (LSE) defined as :

$$\epsilon(\hat{\theta}(n)) = \sum_{j=1}^{n} \| \mathbf{d}(j) - \mathbf{h}(\hat{\theta}(n), \mathbf{i}(j)) \|_2 \, \lambda^{n-j}. \tag{4.4}$$

We conjecture that the following statements are true :

- Switching λ to a value less than 1 results in escaping local minima but often induces explosive divergence.
- Switching λ only for few iterations escapes local minima without inducing explosive divergence. The number of these iterations can be decided from one experimental trial of the specific application at hand.
- When λ is modeled as a function of the error which satisfies these two conditions:
  1. λ is slightly less than 1 when the difference between two consecutive interval average errors is less than a certain threshold.
  2. λ is 1 when the difference defined above exceeds the same threshold chosen above.

An example function is given as follows :

$$\lambda = \begin{cases} \tanh(\alpha * \delta) & \text{if } \delta > \gamma \\ 0.98 & \text{otherwise} \end{cases} \tag{4.5}$$

, **then** this function **guarantees** convergence to satisfactory local minima **if** the parameters (i.e α,L and γ) are selected according to the problem at hand after one experimental trial.

CHAPTER 5

A Different Approach to the Computation of the Gain Matrix

5.1 Motivation

Although the Extended Kalman Filter algorithm to supervised learning is a good alternative to the backpropagation algorithm, some of its drawbacks still make it unattractive for some researchers in the neural network community. One major drawback is a high computational load which we tried to solve by decreasing the number of iterations needed for convergence. Also if we examine the recursion equations needed to update the weights, we note that the algorithm computes the inverse of a matrix whose dimension depends on the dimension of the output. If the dimension of the output is high, the computation of the inverse becomes cumbersome.

In this chapter, we propose a modification that does not require the computation of the inverse. This modification is also aimed toward an analog implementation

5.2 Recomputing the Gain Matrix

The computation of the gain matrix for the global network is equivalent to solving the following equation:

$$-G(n)A^{-1}(n) + \lambda^{-1}P(n-1)H^T(n) = 0, \tag{5.1}$$

where

$$A^{-1}(n) = I + \lambda^{-1}H^T(n)P(n-1)H(n), \tag{5.2}$$

In this equation, we note that instead of conducting a crude computation of the root of the equation, which will require a computation of the matrix A(n), we transform the equation into a continuous time ordinary differential equation (o.d.e.). After convergence, the o.d.e will give a stable solution for the root which can be used as the new value of the gain matrix of each neuron. Consider the associated differential equation for each fixed n

$$-x(t)A^{-1}(n) + \lambda^{-1}P(n-1)H^T(n) = \frac{\partial x(t)}{\partial t}, \tag{5.3}$$

Since for each fixed n, $A^{-1}(n)$ is expected to be positive definite, and hence invertible, the differential equation is a stable linear system. Thus $x_i(t)$ will converge to a constant matrix which we take to be $G(n)$. This way, we avoid the cumbersome computation of the inverse altogether. We observe also that such an o. d. e. is suitable for analog circuit implementation which can be incorporated as a co-processor to a digital computer. In this chapter, our interest is mainly in the (NDEKF) since it was proven to be more efficient than the global (EKF). However, in the NDEKF we will have as many o. d. e. equations to simulate as the total number of neurons in the network not including the input-layer and this is at every update of the weight parameters or at every iteration.

5.3 Simulation Results

Simulations, were conducted to compare the performance of the neuron decoupled algorithm to the proposed modification. In this chapter, we will report the performance on the 4-region classification problem which we have already used on both chapters three and four. The pattern used for this region classification problem was previously described in the earlier chapters. We initialize all parameters including the weights to the same values we chose in the earlier simulations for comparison purposes.

The adaptation of the weights is done as follows:

1. Pick a random point and propagate forward to the output layer.

2. backpropagate the error values (error is computed as the difference between the target output value and the actual computed output) through the whole network.

3. Now do one simultaneous update for all the network weights and matrices. Our modification of the algorithm occurs at this step of the iteration. We do not do a static computation of the gain matrices. Every computation of a gain matrix which involves the computation of an inverse of a matrix, is now a simulation of an o.d.e equation which converges to the gain matrix in few steps. The initial values for the o. d. e. were chosen to be the values of the gain matrices from the previous iteration.

The results of simulations using the same initial condition one will be given in the following table.

The results in this table are very similar to the ones obtained previously with the computation of the inverse. However, these simulations were left training only for 10000 iterations but we still achieved the same performance results. The reason

Table 5.1. The 4-region classification problem with a modified computation of the gain matrix

performance parameters	IC1	IC2	IC3
TAE	0.5872	0.5891	0.5767
IAE	0.5463	0.5420	0.5216
class %	74.6	75.4	76.1
#iterations	10000	10000	10000

behind limiting the training number of iterations is to due to the constraint of time. The o.d.e themselves take few steps to converge inside each training iteration.

In conclusion the computation of the gain matrix using this approach is equivalent to the earlier method that includes the computation of the inverse.

CHAPTER 6

Variable Slopes

6.1 Motivation

Given the considerable success of the linear estimation methods in solving linear problems, researchers have extended these methods to slightly nonlinear problems and sometimes completely nonlinear problems. The "standard" Kalman filter is linear so in order to apply the approach of Kalman filtering to problems of nonlinear type, the Extended Kalman filter was derived. The essential idea of the Extended kalman filter was proposed by Stanley F.Scmidt. It was then named after him as the "Kalman-Scmidt" filter. The Kalman filter is extended mainly by expanding the nonlinear function $h(\theta, i(n))$ describing the model around the current estimate $\hat{\theta}(n-1)$ estimated from all data up to time $(n-1)$. The nonlinearity present in the network is described by the model function $h(\theta, i(n))$ which is the sigmoidal function chosen as $tanh(x)$ in the (FFANN). Due to the chain rule, the derivative of the sigmoidal function is present almost in every element of the gradient matrix $H(n)$. The slope of the sigmoidal function was always fixed at 1.

Even though this function plays a big role in the Extended Kalman filter equations, the influence of the steepness of its slopes on the behavior of the algorithm was never investigated in previous related research. However, in the backpropagation algorithm, the effect of these steepness parameters especially on convergence was well investigated [30]. It was argued in their discussion that a high value of s can make the weights update slowly thus needing a large number of iterations to converge. A too low value of s according to Branko can have these two negative effects.

1. The derivative decreases even in unsaturated regions inducing slow convergence

2. The output of the linear combiner will always fall in the linear region of the sigmoidal function thus loosing any nonlinear effect of the network and dissolving its multilayer structure.

These observations can only suggest that the value of the slope s can not be decided a priori. The solution that was suggested before [30] is to determine an optimum value of s via adaptive means. In the backpropagation algorithm, the slopes of the activation function were updated according to the delta rule [30].

In the case of the Kalman filter algorithm, we propose to investigate the effect of different slopes of the sigmoidal function on the convergence to satisfactory local minima. We will then develop a similar update of the slopes to the one used in the backpropagation algorithm.

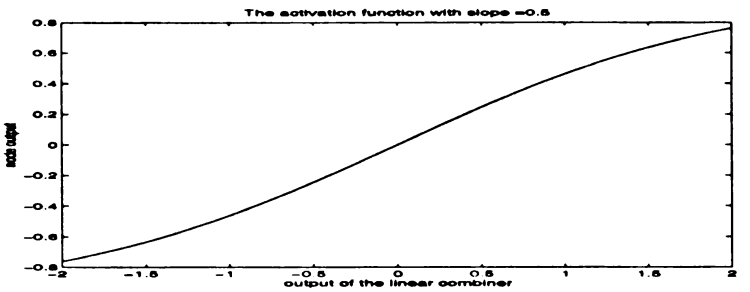


Figure 6.1. The sigmoid function with slope $s = 0.5$

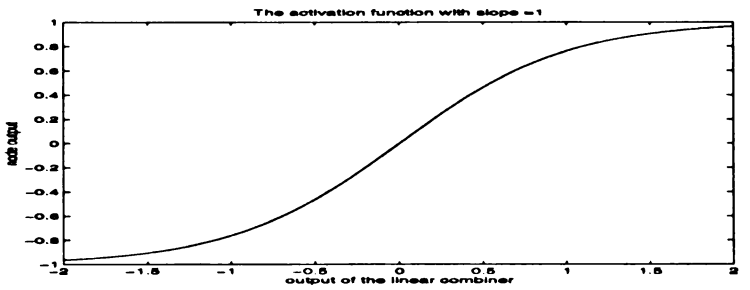


Figure 6.2. The sigmoid function with slope $s = 1$

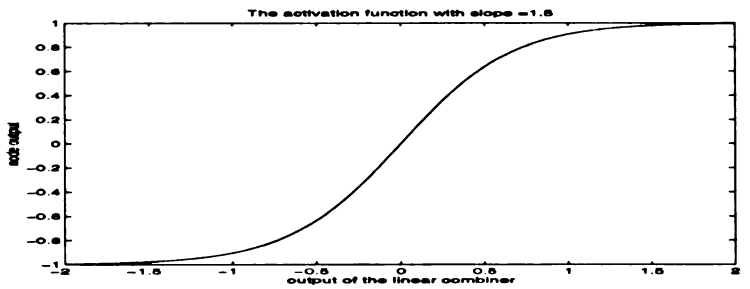


Figure 6.3. The sigmoid function with slope $s = 1.5$

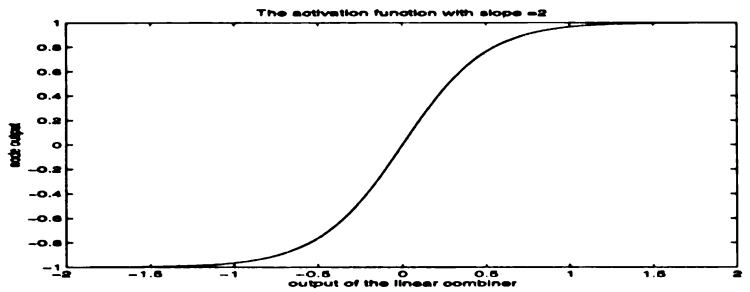


Figure 6.4. The sigmoid function with slope $s = 2$

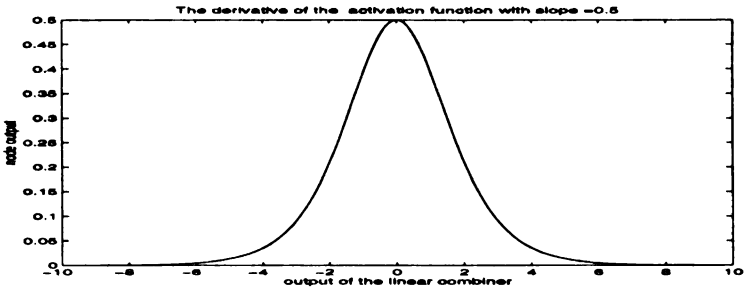


Figure 6.5. The derivative of the sigmoid function with slope $s = 0.5$

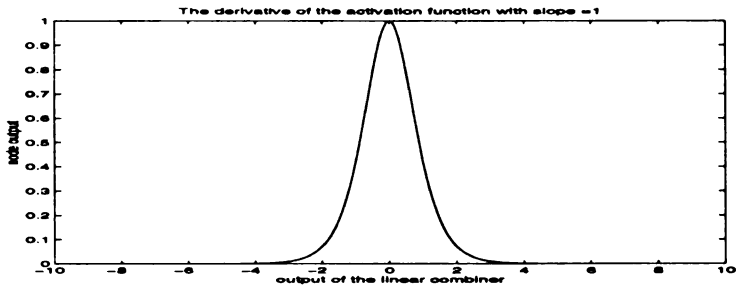


Figure 6.6. The derivative of the sigmoid function with slope $s = 1$

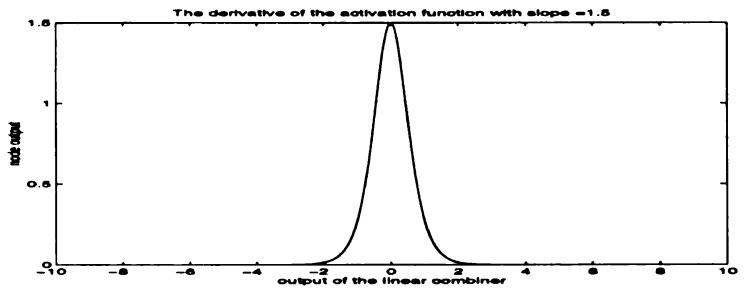


Figure 6.7. The derivative of the sigmoid function with slope $s = 1.5$

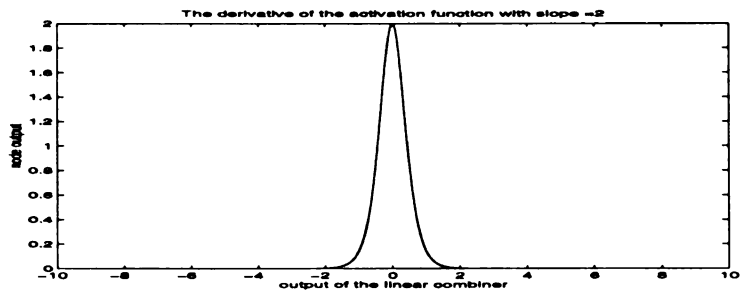


Figure 6.8. The derivative of the sigmoid function with slope $s = 2$

6.2 Preliminary Results

In this section of the chapter, we will illustrate via the 4-region classification problem how using different slopes can have an effect the convergence of the algorithm to satisfactory local minima. The results for each case are tabulated for three different initial conditions.

Table 6.1. The 4-region classification problem with fixed slopes

slope choices		IC1	IC2	IC3
$s = 1$	TAE	0.5757	0.5885	0.5718
	IAE	0.5736	0.5416	0.5295
	class %	76.9	75.1	77.7
	#iterations	50000	50000	50000
$s = 0.5$	TAE	0.6097	0.6282	0.5991
	IAE	0.6054	0.5663	0.5490
	class %	73.5	71.3	76.6
	#iterations	50000	50000	50000
$s = 1.5$	TAE	0.5174	0.5636	0.3875
	IAE	0.5079	0.5178	0.2961
	class %	83.0	79.1	88.9
	#iterations	50000	50000	50000
$s = 2.0$	TAE	0.4306	0.5227	0.3681
	IAE	0.4074	0.4667	0.2946
	class %	86.7	84.0	89.5
	#iterations	50000	50000	50000

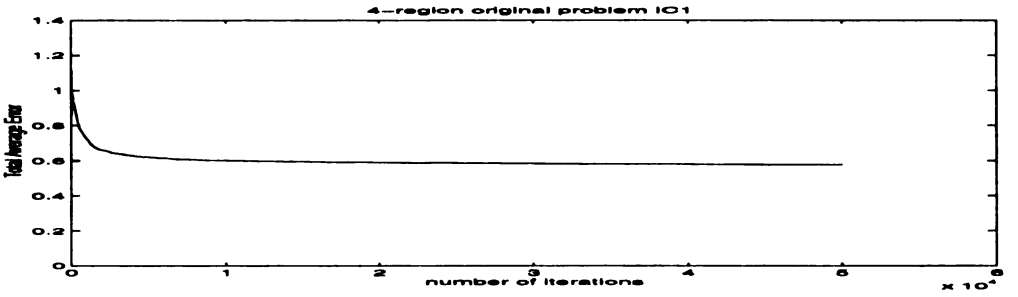


Figure 6.9. The TAE for a fixed slope $s = 1$

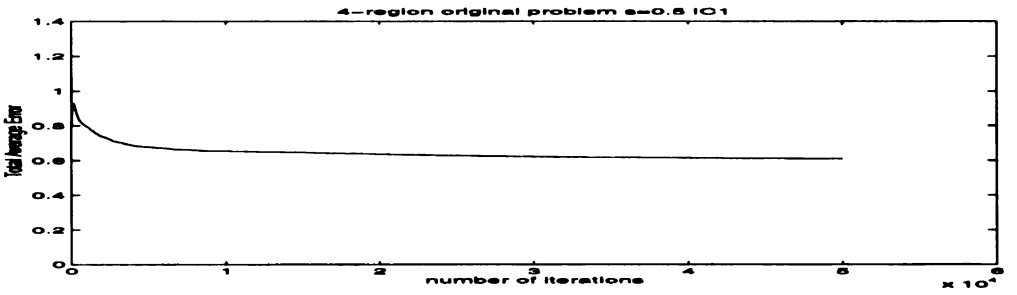


Figure 6.10. The TAE for a fixed slope $s = 0.5$

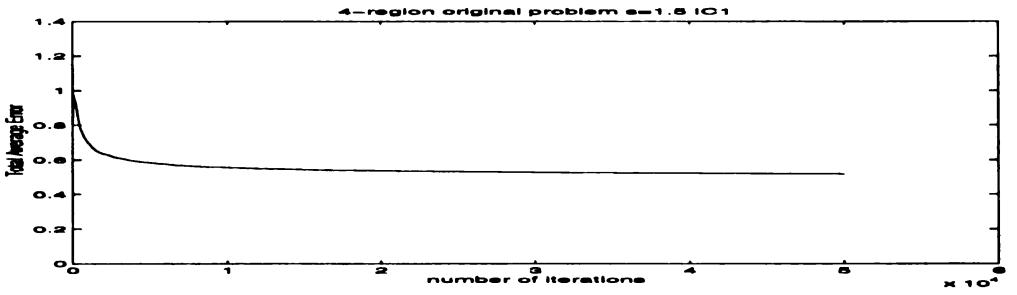


Figure 6.11. The TAE for a fixed slope $s = 1.5$

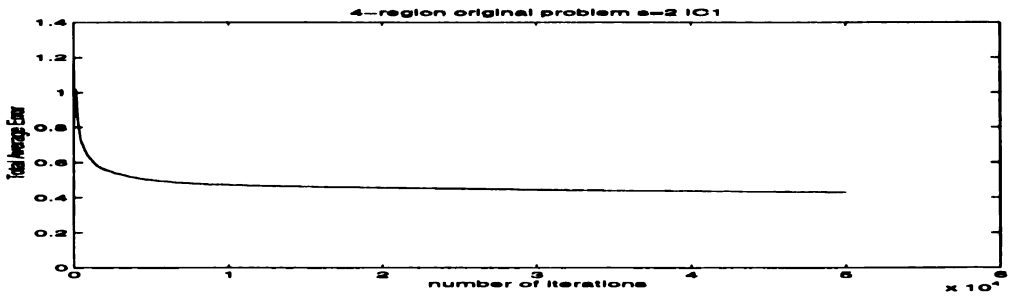


Figure 6.12. The TAE for a fixed slope $s = 2$

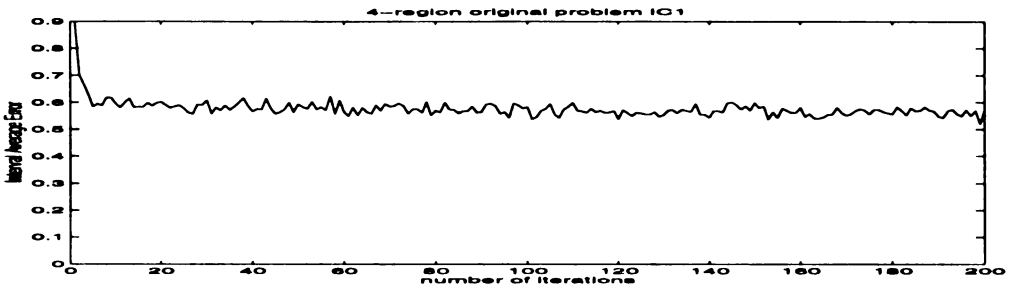


Figure 6.13. The IAE for a fixed slope $s = 1$

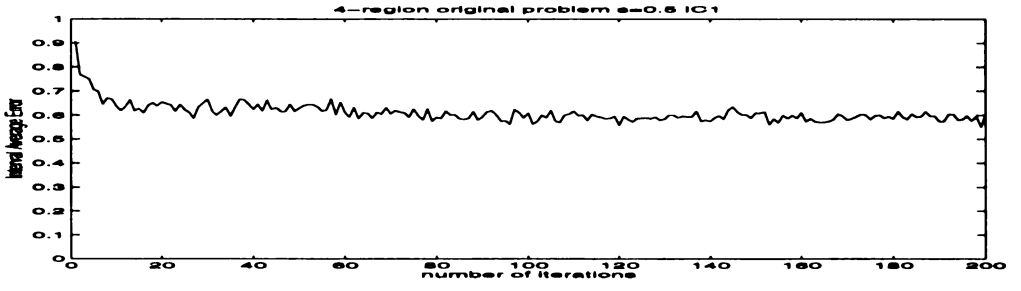


Figure 6.14. The IAE for a fixed slope $s = 0.5$

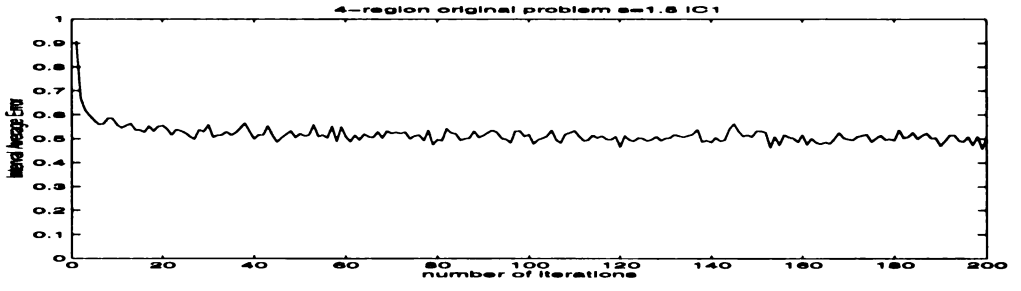


Figure 6.15. The IAE for a fixed slope $s = 1.5$

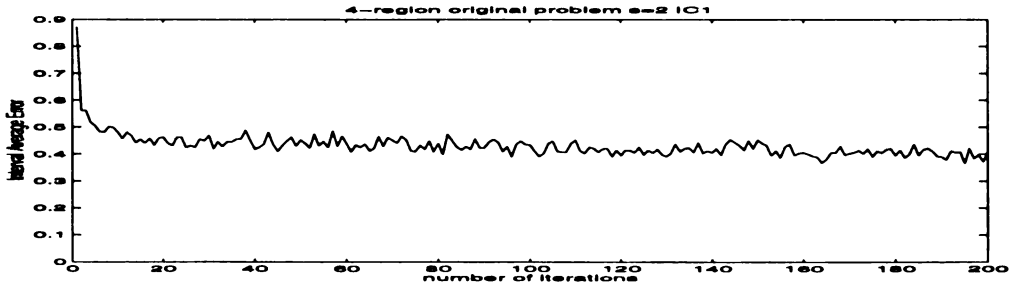


Figure 6.16. The IAE for a fixed slope $s = 2$

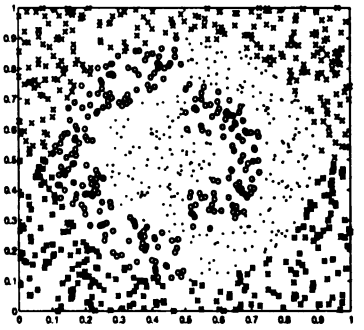


Figure 6.17. The 4-region classification pattern

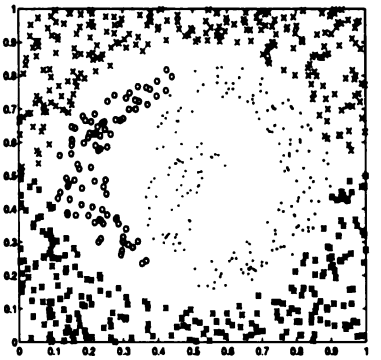


Figure 6.18. Classification Performance for a fixed slope $s = 1$

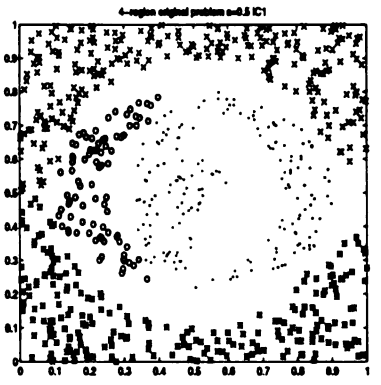


Figure 6.19. Classification Performance for a fixed slope $s = 0.5$

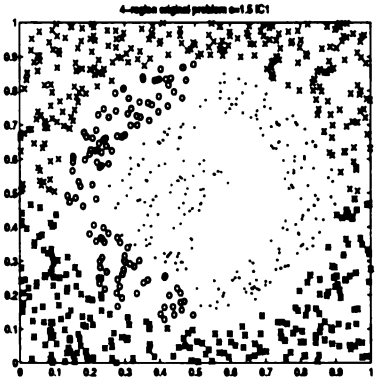


Figure 6.20. Classification Performance for a fixed slope $s = 1.5$

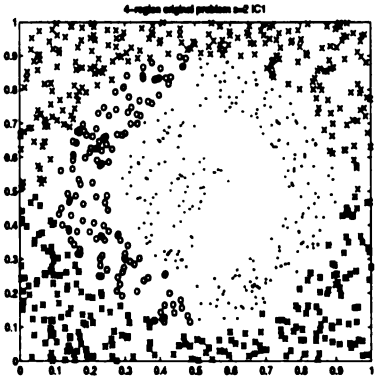


Figure 6.21. Classification Performance for a fixed slope $s = 2$

6.3 The Augmented Recursion Formulas

The (EKF) recursion equations start by a computation of the gain matrix $G(n)$ which is then used to update the weights. The update of the weight parameters is followed by the update of the covariance matix $P(n)$. To these two update equations we add an update equation for the slopes of the sigmoidal function. These slopes are updated at every node of all hidden layers in the network. The update is based on the gradient descent rule. The slope of the nonlinearity s_j^l, where l denotes the hidden layer l and j one of the nodes in that layer, is detemined so as to minimize a certain energy function.

We define the energy function as follows:

$$E_T(n) = 1/2(\| \mathbf{d}(n) - \mathbf{h}(\hat{\theta}(n), \mathbf{i}(n)) \|_2).$$

where n is the time index and $d(n)$ is the vector of desired outputs at time n. $\mathbf{h}(\hat{\theta}(n), \mathbf{i}(n))$ is the nonlinear model function which is a composition of sigmoidal functions.

In this section, without loss of generality, we develop the update equations of the slopes based on the same network architecture used in the 4-region classification problem simulation example. The network is composed of a total of 4 layers including two hidden layers.

Let us call S^2 the vector of slopes for layer 2 and S^3 the vector of slopes for layer 3. The following equations describe the gradients of the energy function with respect to these vectors of slopes.

$$\frac{\partial E_T}{\partial S^2} = (\mathbf{d}(j) - \mathbf{h}(\hat{\theta}(j), \mathbf{i}(j))) \frac{\partial \mathbf{h}(\hat{\theta}(j), \mathbf{i}(j))}{\partial S^2}$$

$$\frac{\partial E_T}{\partial S^2} = (x^4 - x_d^4) \frac{\partial x^4}{\partial S^2}$$

$$\frac{\partial E_T}{\partial S^2} = (x^4 - x_d^4)\frac{\partial x^4}{\partial x^3}\frac{\partial x^3}{\partial x^2}\frac{\partial x^2}{\partial S^2}$$

$$\frac{\partial x^4}{\partial x^3} = w^3$$

$$\frac{\partial x^3}{\partial x^2} = \begin{pmatrix} h'(S_1^2(w_1^2)^T x^2)w_1^2 \\ \vdots \\ h'(S_{10}^2(w_{10}^2)^T x^2)w_{10}^2 \\ 0 \end{pmatrix}$$

$$\frac{\partial x^2}{\partial S^2} = \begin{pmatrix} h'(S_1^2(w_1^1)^T x^1)(w_1^1)^T x^1 \\ \vdots \\ h'(S_{10}^2(w_1^{10})^T x^1)(w_{10}^1)^T x^1 \end{pmatrix}$$

$$\frac{\partial E_T}{\partial S^3} = (x^4 - x_d^4)\frac{\partial x^4}{\partial x^3}\frac{\partial x^3}{\partial S^3}$$

$$\frac{\partial E_T}{\partial S^3} = (x^4 - x_d^4)w^3 \begin{pmatrix} h'(S_1^3(w_1^2)^T x^2)(w_1^2)^T x^2 \\ \vdots \\ h'(S_{10}^3(w_{10}^2)^T x^2)(w_{10}^2)^T x^2 \end{pmatrix}$$

where x^4 is the vector of actual outputs for the output layer and x_d^4 is the vector of desired outputs. x^3 and x^2 are the outputs to layers 2 and 3. x^1 is the input to layer 1.

Also $h(x) = \tanh(sx)$ and $h'(x) = 1/\cosh^2(sx)$

The update equations for the vector of slopes corresponding to layer 2 and 3 are given by the following equations:

$$S^2(n) = S^2(n-1) - \beta\frac{\partial E_T}{\partial S^2} + \rho(S^2(n-1) - S^2(n-2)) \qquad (6.1)$$

$$S^3(n) = S^3(n-1) - \beta\frac{\partial E_T}{\partial S^3} + \rho(S^3(n-1) - S^3(n-2)) \tag{6.2}$$

The augmented recursion formulas are then:

$$A(n) = [I + \sum_{\lambda=1}^{n} H_\lambda p_\lambda(n-1)H_\lambda^T(n)]^{-1}, \tag{6.3}$$

$$G_i(n) = p_i(n-1)H_i^T(n)A(n), \tag{6.4}$$

$$p_i(n) = (I - G_i(n))H_i(n)p_i(n-1), \tag{6.5}$$

$$\theta_i(n) = \theta_i(n-1) + G_i(n)\epsilon(n), \tag{6.6}$$

$$H_i(n) = \frac{\partial f(\theta_1(n-1), ..., \theta_k(n-1), x(n))}{\partial \theta_i(n-1)}, \tag{6.7}$$

$$\epsilon(n) = y_d(n) - f(\theta_1(n-1), ..., \theta_k(n-1), x(n)), \tag{6.8}$$

$$S^2(n) = S^2(n-1) - \beta\frac{\partial E_T}{\partial S^2} + \rho(S^2(n-1) - s^2(n-2)) \tag{6.9}$$

$$S^3(n) = S^3(n-1) - \beta\frac{\partial E_T}{\partial S^3} + \rho(S^3(n-1) - S^3(n-2)) \tag{6.10}$$

6.4 Simulation Results

We conducted the same number of simulations as for the preliminary results. We used the same three initial conditions but since the update of the slopes is being added, we needed to also choose initial conditions for the slope parameters present in the update equations of the slopes. There are four slope initial conditions for each initial condition on the weight parameters.

The results are tabulated in the table and the figures that follow.

Table 6.2. The 4-region classification problem with variable slopes

initial parameters choices		IC1	IC2	IC3
$s_0 = 1\beta = 0.1\rho = 0.001$	TAE	0.5467	0.4019	0.4799
	IAE	0.5013	0.2985	0.3729
	class %	79.3	88.2	84.6
	#iterations	50000	41250	50000
$s_0 = 0.5\beta = 0.1\rho = 0.001$	TAE	0.4010	.4012	0.3265
	IAE	0.2922	0.2958	0.2584
	class %	92.5	86.2	89.7
	#iterations	22250	15500	50000
$s_0 = 1.5\beta = 0.1\rho = 0.001$	TAE	0.4857	0.5444	0.4668
	IAE	0.4692	0.4814	0.3832
	class %	82.4	77.9	85.9
	#iterations	50000	50000	50000
$s_0 = 2.0\beta = 0.1\rho = 0.001$	TAE	0.4617	0.4606	0.3761
	IAE	0.4393	0.3970	0.2998
	class %	82.2	85.6	91.5
	#iterations	50000	50000	6250

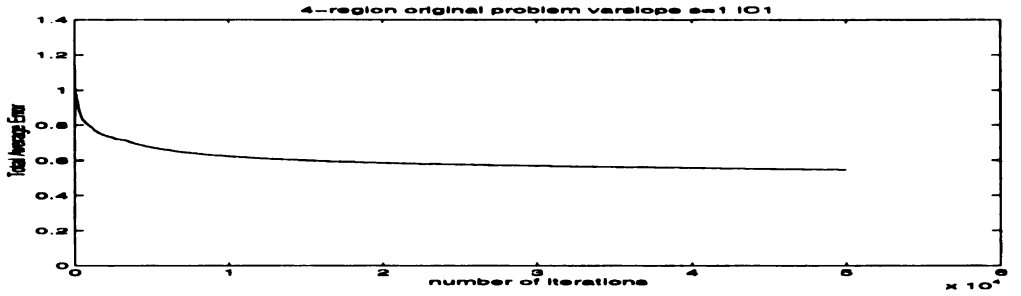


Figure 6.22. The TAE for varying the slope with $s_0 = 1$

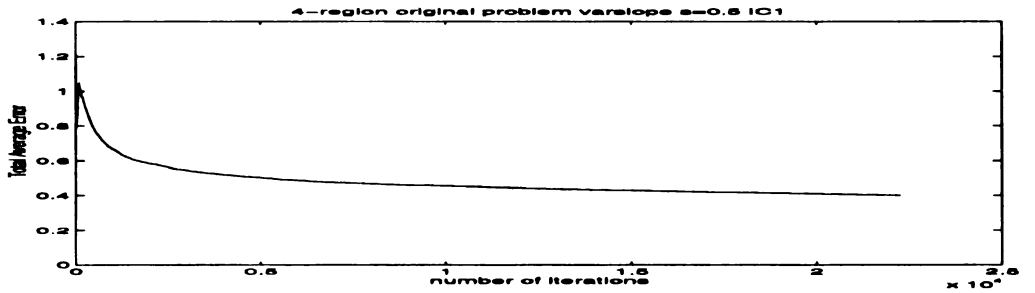


Figure 6.23. The TAE for varying the slope with $s_0 = 0.5$

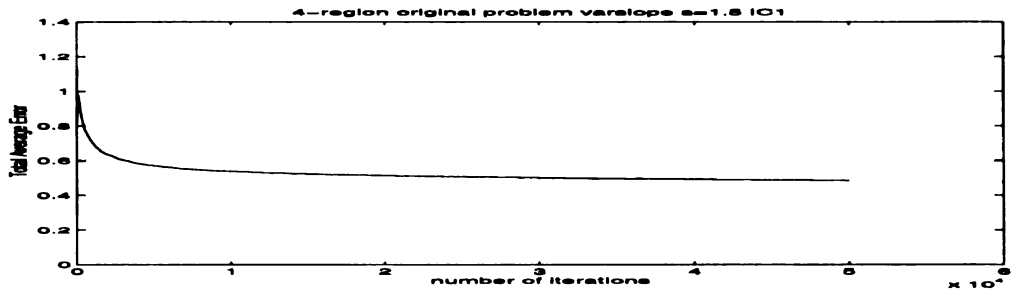


Figure 6.24. The TAE for varying the slope with $s_0 = 1.5$

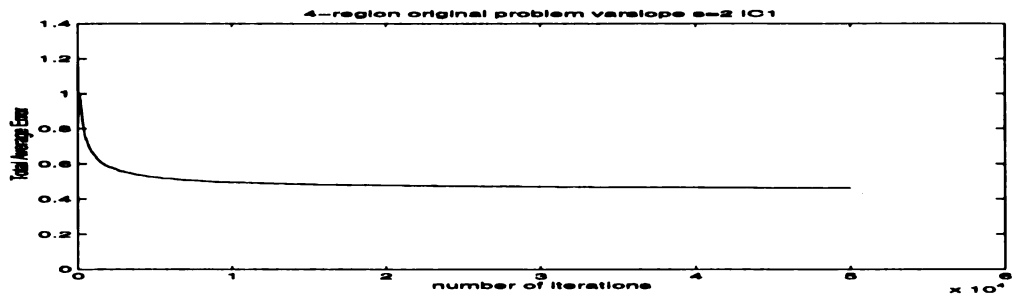


Figure 6.25. The TAE for varying the slope with $s_0 = 2$

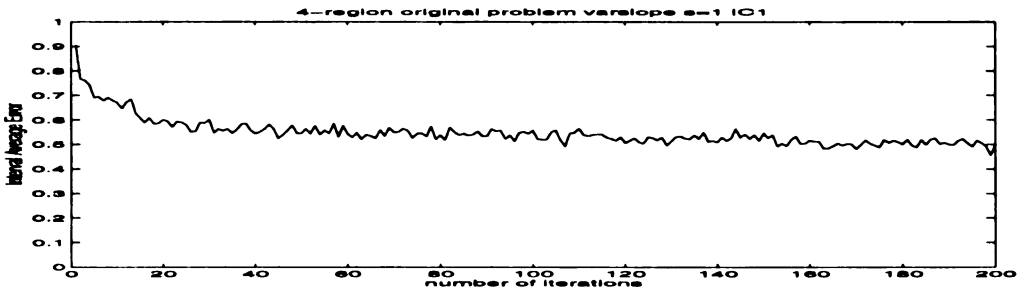


Figure 6.26. The IAE for varying the slope with $s_0 = 1$

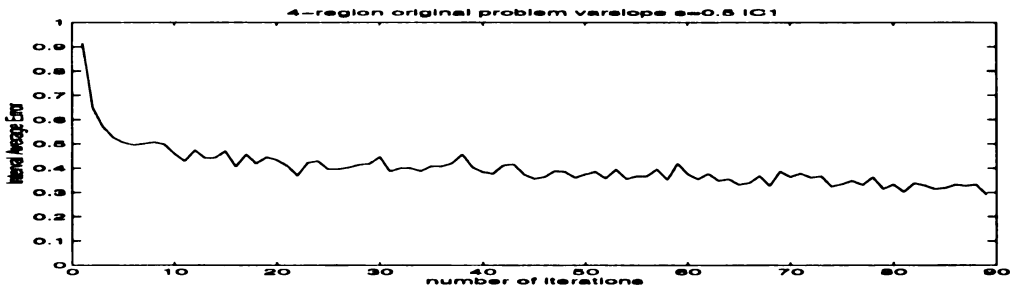


Figure 6.27. The IAE for varying the slope with $s_0 = 0.5$

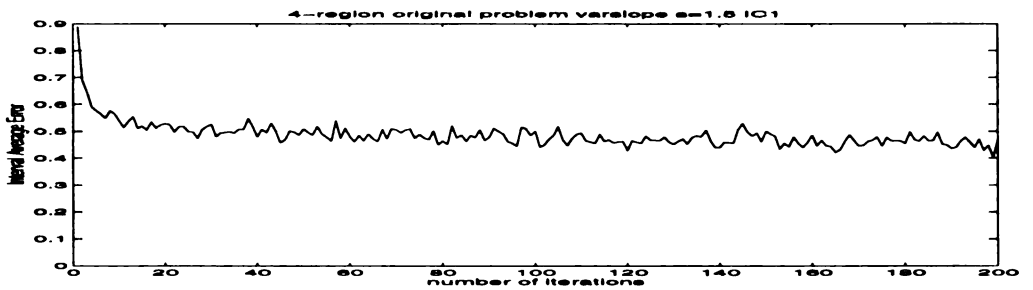


Figure 6.28. The IAE for varying the slope with $s_0 = 1.5$

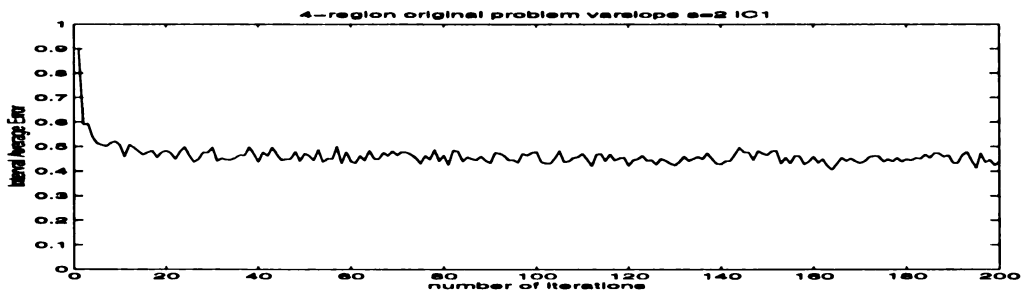


Figure 6.29. The IAE for varying the slope with $s_0 = 2$

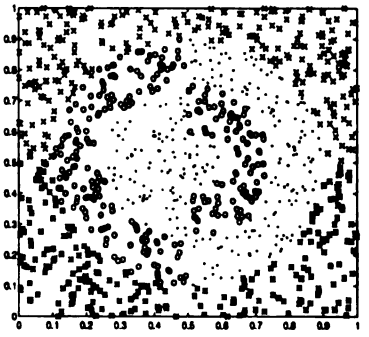


Figure 6.30. The 4-region classification pattern

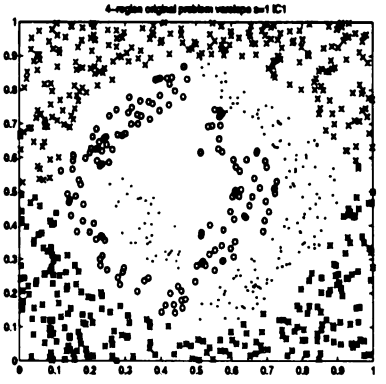


Figure 6.31. Classification Performance for varying the slope with $s_0 = 1$

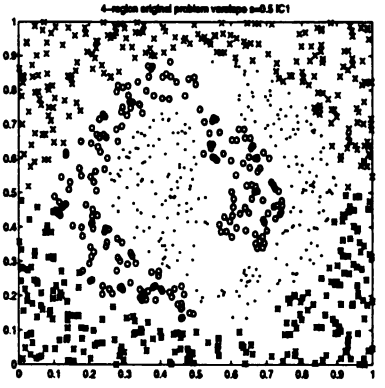


Figure 6.32. Classification Performance for varying the slope with $s_0 = 0.5$

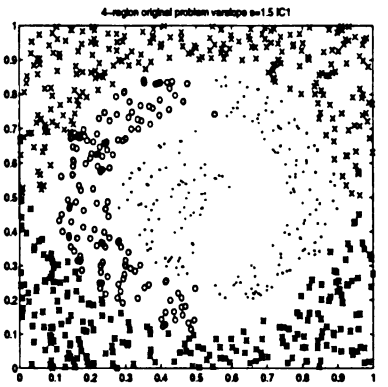


Figure 6.33. Classification Performance for varying the slope with $s_0 = 1.5$

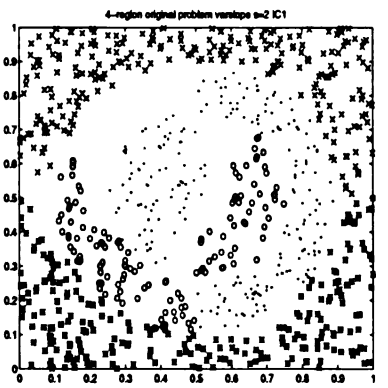


Figure 6.34. Classification Performance for varying the slope with $s_0 = 2$

The numbers in the table suggest that an initial choice of $s_0 = 0.5$ is the best choice amongst all since it is the more efficient in terms of both the performance and the computation load. However, when the slopes were fixed in the previous section, a fixed slope of $s = 2$ was the best especially in terms of performance. In conclusion, if we have no knowledge of what the optimum value of s has to be, which is the case in general, the method of varying the slopes should be used while keeping in mind that the initial choice s_0 should remain in the range of $0.49 < s_0 < 1$. The update of the slopes in conjunction with the previous modifications of the algorithm including the update of the λ parameter is a tremendous improvement of the algorithm altogether.

CHAPTER 7

Summary and Conclusion

Mathematical modeling of real physical systems has been the interest of a considerable number of researchers in the engineering field. Artificial Neural Networks are amongst the most attractive and challenging models these researchers have developed over the years. The strength of these artificial models relies on their ability to mimic natural intelligence. The two main approaches to design these artificial models are those implemented via digital software and those manufactured via analog hardware. This research, however, is designated to the software approach as applied to feedforward artificial neural networks(FFANN). Since supervised learning has been the main learning type used in these networks, the work in this document addressed some of the issues of supervised learning as applicable to FFANNs while using the Extended Kalman Filter approach for training.

7.1 Summary

The research in this thesis was dedicated to a recently introduced (the 1990's) supervised learning algorithm called the Extended Kalman Filter EKF. The more widely used has been the Neuron-Decoupled Extended Kalman Filter (NDEKF) due to its tremendously lower computation load. In the previous reported work this approach has been used in different applications ranging from system identification to nonlinear control. However, the problems that would be faced while practically using the algorithm have never been addressed in any of the previous work.

The lack of any previous discussion of the problems we experimented with while using the algorithm ourselves, was our main motivation for the research in this work. This research has accomplished many tasks that led to the improvement of the algorithm. Starting with our analysis and understanding of how the Extended Kalman Filter algorithm operates, we were able to point out the many problems and obstacles that any user of the algorithm is likely to encounter while using the algorithm. We recall such obstacles as the high computational load and the two forms of divergence that are frequently manifested. The theory behind the algorithm supports most of our findings about the problems and the weaknesses that are present in the algorithm. Also with the help of theory we were able to propose the different modifications aimed toward a more efficient kalman filter training algorithm for supervised learning. These modifications are present in chapters four through six. They range from a suitable update of the energy parameter(λ) to an update of the nonlinearity slopes without adding cumbersome work such as a large number of tunable parameters.

These modifications have the advantage of giving the user the flexibility of adapting the algorithm to any application at hand. Thanks to generalizing the definition of certain functions and parameters the user is able to adjust the variables accordingly with his or her application.

7.2 Conclusion and Future Research

The main contribution in this research is in familiarizing any reader of this thesis with the approach of Extended Kalman Filtering to supervised learning while enabling him or her to decide if the approach with its all advantages and disadvantages is suitable to achieve any research goal he or she may have. The improvements of the algorithm present in this thesis give the user a better more efficient algorithm to work with. The efficiency is improved in terms of both the computation and the performance of the algorithm.

Due to the constraint of time in this research neither all aspects of the algorithm were investigated nor all improvements strategies that were thought of could be implemented. However, future research in the subject is possible and will include trying alternative energy or cost functions that could reduce the convergence time while keeping the performance satisfactory.

APPENDICES

APPENDIX A

Derivation of the Extended Kalman Algorithm Equations

In order to keep this thesis self-contained, the standard derivations for the recursive least squares recursions are included. They are a multidimensional extension of the ones given in [8]

$$\mathbf{\Phi}(n) = \lambda \sum_{j=1}^{n-1} \lambda^{n-1-j} \mathbf{H}(j)\mathbf{H}^T(j) + \mathbf{H}(n)\mathbf{H}^T(n) \tag{A.1}$$

$$\mathbf{\Phi}(n) = \lambda \mathbf{\Phi}(n-1) + \mathbf{H}(n)\mathbf{H}^T(n). \tag{A.2}$$

$$\mathbf{r}(n) = \lambda \mathbf{r}(n-1) + \mathbf{H}(n)(\mathbf{d}(n) - \xi(n)) \tag{A.3}$$

Using the matrix inversion lemma:

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{C}^T \tag{A.4}$$

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{I} + \mathbf{C}^T\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{B}. \tag{A.5}$$

with $\mathbf{B} = \mathbf{\Phi}^{-1}(n-1)\lambda^{-1}$ and $\mathbf{C} = \mathbf{H}(n)$, we get

$$\begin{aligned}\mathbf{\Phi}^{-1}(n) &= \lambda^{-1}\mathbf{\Phi}^{-1}(n-1) - \lambda^{-2}\mathbf{\Phi}^{-1}(n-1)\mathbf{H}(n) \\ & X[I + \lambda^{-1}\mathbf{H}^T(n)\mathbf{\Phi}^{-1}(n-1)\mathbf{H}(n)]^{-1} \\ & X\mathbf{H}^T(n)\mathbf{\Phi}^{-1}(n-1), \end{aligned} \tag{A.6}$$

Defining $\mathbf{P}(n) = \mathbf{\Phi}^{-1}(n-1)$, and

$$\mathbf{G}(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{H}(n)[\mathbf{I} + \lambda^{-1}\mathbf{H}^T(n)\mathbf{P}(n-1)\mathbf{H}(n)]^{-1}, \tag{A.7}$$

we can rewrite eqn(A6) as

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{G}(n)\mathbf{H}^T(n)\mathbf{P}(n-1). \tag{A.8}$$

Equation (A7) can be rewritten as

$$\mathbf{G}(n) + \lambda^{-1}\mathbf{G}(n)\mathbf{H}^T(n)\mathbf{P}(n-1)\mathbf{H}(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{H}(n), \tag{A.9}$$

or

$$\mathbf{G}(n) = [\lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{G}(n)\mathbf{H}^T(n)\mathbf{P}(n-1)]\mathbf{H}(n). \tag{A.10}$$

Therefore the Kalman gain can be expressed as

$$\mathbf{G}(n) = \mathbf{P}(n)\mathbf{H}(n), \tag{A.11}$$

The parameter estimate for the filter weight $\hat{\theta})(n)$ can then be expressed as

$$\hat{\theta}(n) = \mathbf{P}(n)\mathbf{r}(n) \tag{A.12}$$

$$\hat{\theta}(n) = \mathbf{P}(n)\lambda\mathbf{r}(n-1) + \mathbf{P}(n)\mathbf{H}(n)(\mathbf{d}(n) - \xi(n)). \tag{A.13}$$

Substituting (A8) in the first term of (A12) we get

$$\begin{aligned} \hat{\theta}(n) &= \mathbf{P}(n-1)\mathbf{r}(n-1) - \mathbf{G}(n)\mathbf{H}^T(n)X \\ & \quad \mathbf{P}(n-1)\mathbf{r}(n-1) + \mathbf{P}(n)\mathbf{H}(n)(\mathbf{d}(n) - \xi(n)). \end{aligned} \tag{A.14}$$

From (A11) we get

$$\hat{\theta}(n) = \hat{\theta}(n-1) + \mathbf{G}(n)(\mathbf{d}(n) - \xi(n) - \mathbf{H}^T(n)\hat{\theta}(n-1)), \tag{A.15}$$

or

$$\hat{\theta}(n) = \hat{\theta}(n-1) + \mathbf{G}(n)(\mathbf{d}(n) - \mathbf{h}(\hat{\theta}(n-1), \mathbf{i}(n))). \tag{A.16}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] T. Hrycej, *Modular Learning in Neural Networks.* Toronto, Canada: John Wiley and Sons, Inc., 1992.

[2] P. D. Wasserman, *Advanced Methods in Neural Computing.* London, England: International Thomson Publishing, 1993.

[3] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, pp. 295–307, 1988.

[4] T. Samad, "Backpropagation improvements based on heuristic arguments," *Proceedings of International Joint Conference On Neural Networks*, pp. 565–568, Washington, DC: IEEE Press 1990.

[5] M. Ahmad, *Supervised Learning of Feedforward Artificial Neural Networks Using Different Energy Functions.* Michigan State University, East Lansing, Mi: Michigan State University, Dec 1991.

[6] R. W. Prager and Fallside, "The modified Kanerva Model for Automatic Speech Recognition," *In IEEE workshop on Speech Recognition*, vol. Arden House, Harriman NY, May 1988.

[7] B. Irie and S. Miyake, "Capabilities of Three-layered Perceptrons," *Proceedings of the IEEE International Conference on Neural Networks*, vol. 1, pp. 641–648, June 1988.

[8] S. Haykin, *Adaptive filter theory.* NJ:Prentice Hall: Englewood Cliffs, 1991.

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

[10] F. Rosenblatt, *Principles of Neurodynamics.* Washington, DC: Spartan Books, 1961.

[11] S. Grossberg, "Adaptive pattern classification and universal recoding I: Parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121–134, 1976.

[12] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man and Cybernetics* **SMC**, vol. 18, pp. 49–60, 1987.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by error back propagation in parallel distributed processing: Explorations in the microstructures of cognition," *MA:MIT Press. Cambridge*, vol. 1, pp. 318–362, 1986.

[14] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, pp. 267–273, 1982.

[15] T. Kohonen, *Self-Organization and Associative Memory.* New York: Springer-Verlag, 1988.

[16] Hecht-Neilson, "Counterpropagation Networks," *Proceedings of the IEEE First International Conference on Neural Networks*, vol. 2, pp. 19–32, 1987.

[17] B. Widrow, "An adaptive "Adaline" neuron using chemical "Memistors"," *Technical Report*, vol. 1553-2, October 1960.

[18] B. Widrow and F. W. Smith, "Pattern recognizing control systems," *Computer and Information Sciences Symposium Proceedings*, vol. Sparton Books, Washington D. C. 1963.

[19] M. S. Grewal, *Kalman filtering : theory and practice.* N. J. : Prentice-Hall: Englewood Cliffs, 1993.

[20] H. W. Sorenson, "Kalman Filtering Techniques," *In Advances in Control Systems Theory and Applications*, vol. 3, pp. 219–292, ed. C. T. Leondes. New York: Academic Pres 1966.

[21] S. Singhal and Wu, "Training Multilayer Perceptrons with the Extended Kalman Algorithm," *In Advances in Neural Information Processing Systems*, pp. 133–140, Denver 1988, 1ed. D. S. Touretsky, San Mateo, CA: Morgan Kaufmann 1988.

[22] S. A. Shah and F. Palmieri, "MEKA - A Fast, Local Algorithm for Training Feedforward Neural Networks," *In International Joint Conference on Neural Networks*, vol. 3, pp. 41–45, NewYork: IEEE 1990.

[23] F. Palmieri and S. Shah, "Fast training of multilayer perceptron using multilinear parameterization," *Proceedings of International Joint Conference on Neural Networks*, vol. 1, pp. 696–699, IEEE Press 1990.

[24] K. S. Narendra and K. Parasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, pp. 4–27, 1990.

[25] L. Jin, P. N. Nikiforuck, and M. M. Gupta, "Decoupled Recursive Estimation Training and Trainable Degree of Feedforward Neural Networks," *IJCNN*, 1991.

[26] S. Shah, F. Palmeiri, and M. Datum, "Optimal Filtering Algorithms for Fast Learning in Feedforward Neural Networks," *Neural Networks*, vol. 5, pp. 779–787, 1992.

[27] G. V. Puskorius and L. A. Feldkamp, "Decoupled Extended Kalman Filter Training of Feedforward Layered Networks," *In IEEE International Conference on Neural Networks*, vol. 1, pp. 771–777, 1991.

[28] G. E. Bottomley and S. T. Alexander, "A Theoretical Basis for the Divergence of Conventional Recursive Least Squares Filters," , 1989.

[29] G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *In IEEE Transactions on Neural Networks*, vol. 5, pp. 279–297, March 1994.

[30] B. Soucek and the Iris Group, *Neural and intelligent systems integration.* Newyork- Brisbane- Toronto- Singapore: A wiley Interscience Publication, John Wiley Sons, Inc., 1991.