Spring 1998

Stability and weight smoothing in CMAC neural networks

David Paul Campagna
University of New Hampshire, Durham

Stability and Weight Smoothing in CMAC Neural Networks

by

David Paul Campagna

BSEE University of New Hampshire, 1986
MSEE University of New Hampshire, 1989

Dissertation

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

in

Engineering

May, 1998

This dissertation has been examined and approved.

Dissertation Director, Dr. L. Gordon Kraft
Professor of Electrical Engineering

Dr. W. Thomas Miller III
Professor of Electrical Engineering

Dr. Filson Glanz
Professor Emeritus of Electrical Engineering

Dr. Michael J. Carter
Associate Professor of Electrical Engineering

Dr. Barry K. Fussell
Associate Professor of Mechanical Engineering

Dr. Donald W. Hadwin
Professor of Mathematics

Date

Dedication

This dissertation is dedicated to my grandfather, Richard J. Campagna. Pepère, you nurtured my interests in math and science right from the beginning. You helped me to learn how to ask questions and to wonder why things are the way they are. I will always remember with love and gratitude the many projects and experiments we did. The genetic study of the soldier bean, model rockets, my first chemistry set, learning to play cribbage, rocks and minerals, the electromagnet, my first Heathkit project, the slide rule, DC electronics, blue potatoes and popcorn. If I thought I was interested in it, you'd help me look into it more. Thank you, Pepère. I love you and I miss you.

Acknowledgments

This dissertation would not have become a reality without the help and support of a great many people. I would like to express my deepest love and gratitude to my parents, Paul and Therese Campagna, for their constant support and encouragement and for setting such a wonderful life example. To Dr. Gordon Kraft, teacher, advisor, and friend, my heartfelt thanks for not giving up on me. Lord knows I didn't make it easy, but you were always there. To my committee, thank you for your willingness to work with me when this dissertation suddenly reappeared from the mists. To Dr. Robert P. Hewes, you're "top shelf" in my book. Some day we will get to work together again and "...woe to the wicked." To Brian Box, Ben Brown, Gerry Jankauskus, Jake Freedman, Steve Scalera, Allen Johnson, Steve Neill, I don't see how I'd have made it here without you. Thank you to my management at Sanders for being willing to accommodate me in this undertaking. To my siblings, family and friends, success is about the people in your life and, thanks to you, my life will always be a success. To Sarah Robyn Thee, my Best Friend and Life Partner, my wife-to-be, a million thankyou's with lots of spray on them. You've saved my life and given so much to support me on this path. Without you this dissertation probably wouldn't have happened, so it's yours as much as mine. Thanks you for staying. This work was supported in part by a grant from the National Science Foundation (IRI-9112531).

Table of Contents

List of Tables

List of Figures

Abstract

Stability and Weight Smoothing in CMAC Neural Networks

by

David Paul Campagna

University of New Hampshire, May, 1998

Although the CMAC (Cerebellar Model Articulation Controller) neural network has been successfully used in control systems for many years, its property of local generalization, the availability of trained information for network responses at adjacent untrained locations, although responsible for the networks rapid learning and efficient implementation, results in network responses that is, when trained with sparse or widely spaced training data, spiky in nature even when the underlying function being learned is quite smooth. Since the derivative of such a network response can vary widely, the CMAC's usefulness for solving optimization problems as well as for certain other control system applications can be severely limited. This dissertation presents the CMAC algorithm in sufficient detail to explore its strengths and weaknesses. Its properties of information generalization and storage are discussed and comparisons are made with other neural network algorithms and with other adaptive control algorithms. A synopsis of the development of the fields of neural networks and adaptive control is included to lend historical perspective. A stability analysis of the CMAC algorithm for open-loop function learning is developed. This stability analysis casts the function learning problem as a unique implementation of the model ref-

erence structure and develops a Lyapunov function to prove convergence of the CMAC to the target model. A new CMAC learning rule is developed by treating the CMAC as a set of simultaneous equations in a constrained optimization problem and making appropriate choices for the weight penalty matrix in the cost equation. This dissertation then presents a new CMAC learning algorithm which has the property of "weight smoothing" to improve generalization, function approximation in partially trained networks and the partial derivatives of learned functions. This new learning algorithm is significant in that it derives from an optimum solution and demonstrates a dramatic performance improvement for function learning in the presence of widely spaced training data. Developed from a completely unique analytical direction, this algorithm represents a coupling and extension of single- and multi-resolution CMAC algorithms developed by other researchers. The insights derived from the analysis of the optimum solution and the resulting new learning rules are discussed and suggestions for future work are presented.

Introduction

In this dissertation the CMAC (Cerebellar Model Articulation Controller or Cerebellar Model Arithmetic Computer) neural network is modeled in matrix form and its weight adaptation, or learning, process is structured as a constrained optimization problem. Properties of the resulting optimized weight update law are extracted for application in a modified CMAC weight adaptation law. The motivation for this effort is derived from the desire to improve the performance of the CMAC in control system applications. The new weight adaptation law will accomplish this performance enhancement by improving the function approximation capability of the CMAC network.

The idea of 'control' is a familiar one with connotations of forcing a person or system to perform in some constrained way under the direction of and to the specifications of a 'controller.' The most common neural controller is the brain. The brain controls the complex kinematics and dynamics of the body enabling all living creatures to perform physical activities using their bodies. Human bodies are extremely complicated biomechanical systems with over 600 force generating actuators (muscles) connected to a structural framework with 200 independent segments (bones) with feedback provided by millions of sensors (nerves). All aspects of body control are also nonlinear meaning that the range of responses to stimuli are not simply scaled, filtered versions of the original stimuli. Man-made neural controllers attempt to mimic the capabilities of the brain with simple networks containing a large number of adjustable parameters. The range of control systems will be explored in the following paragraphs, beginning with classical, linear con-

trollers and progressing to nonlinear and adaptive controllers and finally to neural controllers. The CMAC neural network, that is the topic of study for this research, is often used in control systems which classify as neural control systems.

A control system is typically an electrical and/or mechanical system whose purpose is to cause some other system, referred to as "the plant," to perform in some prespecified or desired way. A classic example of a control system is the cruise control found in many cars. The desired behavior is for the car to travel at the speed set by the driver. The cruise control receives as input the vehicle speed, compares this speed to the desired speed set by the driver, determines an error signal from this comparison, and changes the throttle in order to make this error as small as possible. This is a feedback error controller since the control signal (throttle position) is determined by the error between a desired value and the measured plant output.

The classical design of such a cruise control might be of the PID (Proportional, Integral, Derivative) variety. There are three fixed parameters, called gains, in a PID controller which determine the behavior of this system. They are the three system gains K_p, K_i, and K_d. The choice of these parameters to achieve a particular type of behavior depends upon the properties of the plant being controlled. Armed with enough information about the plant, there are very well established design methodologies available to design a controller to exacting specifications for an "ideal, linear system." In reality, however, few systems are truly linear. Fortunately, many are linear enough for this class of controllers to provide quite acceptable performance.

In many cases, full knowledge of the plant is not available. If the plant is linear, there are still several well developed methodologies available to the control system

designer. Typically a mathematical form is assumed for a model of the plant and a controller is designed with several adjustable parameters. Since the parameter values used in the controller depend upon the values in the plant model, accurate determination of the plant properties is essential. Often these parameter values can be determined mathematically from a study of the plant structure or experimentally. More sophisticated methods such as least squares are available when some or all of the parameters cannot be determined by analysis or direct measurement.

For observable plants, where the internal properties of the plant can be determined from outputs or measurable states, the model parameters can be derived from sufficient measurement data in a process known as parameter identification. Once the identification procedure has converged to stable parameter values, the model is considered complete and a controller is designed to it. This controller is then used to control the actual plant. The performance of the controller is directly related to the quality of the model and its identified parameters.

In some cases the controller itself contains adjustable parameters. The values of these parameters are adjusted by an adaptation law while the controller is in operation. The adaptation algorithm is usually attempting to estimate postulated plant parameters for the mathematical model so that these estimates may be used in the control system. This process is referred to as simultaneous system identification and control. If plant parameters are estimated and these estimates are used in the controller, the process is called indirect control. If the parameters in the controller are adjusted without estimating the plant parameters, the process is called direct control. For linear systems this theory is well developed and includes detailed stability analyses. Two different approaches to this adap-

tive control design are the Model Reference Adaptive Control (MRAC) strategy and the Self Tuning Regulator (STR) strategy. More will be said about these approaches in the Adaptive Control Section 1.4 of Chapter 1. It is usually important to keep the number of adjustable parameters small in order to keep the computational requirements manageable for real-time control applications. These techniques can break down if the plant is significantly nonlinear.

For plants which are known not to be linear it is often possible to linearize the mathematical model about one or more operating points and still successfully apply the methods described above. A common example of a nonlinear system (and an unstable one too) is the inverted pendulum. Often referred to as the broom balancing problem, the goal is to balance the broom on the palm of one's hand. The inverted pendulum, in this case the broom, is a very popular benchmark problem in control systems since inverted pendulums are easy to build, relatively easy to model, easy to linearize about the balance point, but difficult to control well. The difficulty lies in the fact that the control effort needed to right the pendulum increases very rapidly with the deviation from vertical. In addition this plant is inherently unstable. The broom will fall over no matter how carefully the initial position is set (ignoring friction and other effects).

It is possible to linearize the inverted pendulum about the balance point but system performance is quickly degraded when the state of the plant is different than the operating point about which the model was linearized. Sometimes even small deviations from the operating point can lead to system instability.

One possible solution to this problem might be to choose more operating points and switch from one controller to the next as the system moves from operating point to

operating point. This would involve many controllers, possibly even adaptive controllers, and many parameters to manage. Another approach that is gaining popularity as the number of successful applications increases is to use a neural network as all or part of the controller. Neural networks can have thousands or even millions of adjustable parameters, called weights. The weights are adjusted by an adaptation law or learning rule specific to the particular network. The advantage of using neural networks is that they are capable of managing even the nonlinear aspects of the plant. There are mathematical proofs [187] for several network architectures stating that these types of networks can theoretically represent any kind of function. There are, unfortunately, almost no results suggesting how to construct a network for a particular task and then train it to represent the desired function. So, while it is comforting to know that a solution exists, knowing whether the particular network being used can actually solve the problem is still left to trial and error.

The important features of neural networks is that they are capable of representing and storing both linear and nonlinear functions and they are capable of adjusting these stored functions in response to new training or performance information. These properties allow neural network based controllers to span the range from linear to nonlinear and adaptive controllers. They are often used to augment more traditional controllers. For example, a system might be stabilized with a linear controller while the neural network based controller learns system nonlinearitities and augments the linear controller outputs to compensate for system properties not accounted for in the primary controller.

It is important to note that there are many different types of neural networks. The historical overview of the next chapter will highlight the major network architectures. For now it is sufficient to note that each network architecture has certain properties that deter-

mine the class of problems to which it can be applied. Self-organizing networks are good at extracting hidden patterns from data. Multi-layer perceptrons are good at determining global maps. Associative memories are good at building up a map from local information. When considering a network architecture for use in a control system, the concerns are: the speed at which learning occurs, the stability of the learning algorithm, the stability of the closed-loop system including the controller, the accuracy of the learned function, and often quality of the derivative(s) of the learned function as well.

The focus of this dissertation is a network of the associative memory type: CMAC. The CMAC has been successfully used in control systems at the University of New Hampshire since the mid 1980's and elsewhere since the early 1980's. Advantages of the CMAC include rapid learning convergence and the availability of dedicated hardware accelerators. The most attractive property of the CMAC algorithm, however, is that it can be easily implemented on even low-end, general purpose hardware like PC's and still operate rapidly enough to provide real-time control of complex systems like the General Electric P5 five axis industrial robot. The CMAC networks learns orders of magnitude faster than a comparable multi-layer perceptron network trained with the backpropagation algorithm. Some disadvantages of the CMAC include that it is not a universal function approximator [23] like the multi-layer perceptron and the radial basis function networks. That is, there exists a family of functions (quantized analog or digital) whose internal representation is orthogonal to the representational basis in the weight space and therefore cannot be represented by the CMAC at all. Despite this theoretical limitation, the CMAC has been shown capable of learning many of the functions required for control applications.

CMAC is a local learning network which has both advantages and disadvantages. The local learning property contributes to the remarkable speed of the CMAC algorithm since any access to the network for learning or for retrieving information only requires that some small number (typically less than 256) of the network weights must actually be accessed. This property can also contribute to particularly rapid learning convergence when learning periodic and well sampled functions. In contrast, global learning networks like multi-layer perceptron networks require that all the weights be adjusted for training and all the weights can potentially contribute to the network response for any input.

Problems with the local learning property appear primarily when the CMAC network is only partially trained. If the training points are close enough together then the network generalization, its ability to share information among related input states, will enable the network to give reasonable responses at points in the vicinity of trained points where training has not actually occurred. The particular difficulty in control system applications and especially for optimization type problems is that the derivative of the network output can often be grossly in error even when the network output itself is acceptable. One solution to this problem is to train the function derivative into the network as an additional output. This increases the complexity of the system by essentially requiring a completely new network to learn the derivative. Additionally, a measured or estimated derivative may be corrupted by considerably more noise than the non-differentiated signal. This can make learning more difficult although the learning process in CMAC is inherently a low-pass operation.

Another more insidious problem with the existing learning algorithm for CMAC is that it allows some of the weight values to grow without bound. This poses less of a prob-

lem in the theoretical treatment than it does in application. In the case where CMAC weights are represented by 16-bit integers, for example, it doesn't take too much attention from the learning algorithm to cause a weight value to grow beyond that which can be represented by 16 bits. As a result, a weight value can suddenly change from a large positive number to an equally large negative number or from a large negative number to a large positive number. The effect of such a sign change can be catastrophic for the output(s) to which that weight contributes. Indeed, the effects can ripple throughout the network. A weight magnitude monitor has been added to the UNH CMAC algorithm as a fix. For each set of weights accessed during a training cycle, the weights are also adjusted so that deviations from the set average value are kept small. This allows the weight structure to be trained in while preventing runaway weights.

The new learning rule developed in this dissertation specifically addresses the above problems. The new rule implements weight smoothing and automatic magnitude control thereby improving the network function approximation, the behavior of the differentiated network output, the network generalization and the overall stability of the network weights. The impact of this new algorithm on CMAC learning speed and quality are examined and examples are given.

Chapter 1 presents an historical perspective on the development of both the neural network field and the adaptive control field to provide some context for the discussion of CMAC. Performance comparisons are drawn from the literature to highlight CMAC characteristics and motivate its use in control systems. Chapter 2 covers the detailed information about the CMAC algorithm, documenting the evolution of the algorithm since its original development in 1972. Chapter 3 addresses the issue of learning convergence and

stability for the CMAC algorithm. Chapter 4 lays the groundwork for developing the new weight adjustment law. The CMAC is modeled as a set of simultaneous equations and the solution is determined by solving a constrained optimization problem in the CMAC weights. Chapter 5 explores the requirements derived from the matrix rule developed in Chapter 4. In particular, the CMAC generalization property is studied and a modified generalization technique is developed in order to support the new weight adjustment law. A crude first attempt at implementing the new adjustment law in actual CMAC code is presented along with experimental results demonstrating the performance of this approximation to the new weight update rule. Comparisons with other update rules are provided. Chapter 6 summarizes the work presented in this dissertation and draws conclusions about the results. Suggestions for future work are also presented.

Chapter 1

Background

1.1 What are Neural Networks

For the purposes of discussion in this dissertation, a neural network is defined to be any computer algorithm, mathematical model, or piece of hardware whose structure is either a) modeled, in whole or in part, after the connectivity and physical functioning of the neurons in the human brain, or b) whose structure is intended to duplicate the functioning of some cognitive unit of the human brain without paying specific attention to properties of biological neurons. It is worth noting at this point that the neural network field is not the same as the field of study known as AI or Artificial Intelligence. The term AI refers to the field of symbolic knowledge representation where computer programs manipulate some knowledge base in an effort to mimic the high-level decision making and reasoning capabilities of the human brain/mind. The boundary between neural networks and AI is ill-defined at best since one can certainly argue that neural networks form some abstract representation of the problem they are trained to solve. Based upon where neural networks find application, it seems more appropriate to refer to them as low-level cognitive blocks used primarily for pattern recognition and function learning and to refer to AI for high-level cognitive blocks used in applications like expert systems where large amounts of information recognizable to humans as knowledge are manipulated and pre-

sented in a way that is useful and helpful to users of that system.

1.2 A Brief History of Neural Networks

The field of neural networks began roughly in 1943 when Warren McCulloch and Walter Pitts developed the first model for a computing element based on some of the properties of biological neurons[93]. Much work followed demonstrating how networks formed of interconnected McCulloch-Pitts neurons could perform many useful functions. In 1949 Donald Hebb [62] presented a theory for learning suggesting that information could be stored in connections between individual neural computing elements and presented a learning rule, known as the Hebbian learning rule, to perform this connection update. This was the beginning of "machine learning."

The first neural network machine was built by Marvin Minski in 1951 and consisted of 40 neural computing elements with adjustable interconnections. At the same time, researchers were developing a fledgling theory involving adjustable parameters in automatic control systems that would grow to become the powerful field of adaptive controls. In 1958 the neural computing element known as the "perceptron" was developed by Frank Rosenblatt[136].

The early sixties were a time of powerful developments in the field of neural networks. The development of the ADaptive LINEar combiner (ADALINE), the Widrow-Hoff learning rule to train it, and later the MADALINE (for Many ADELINEs) by Bernard Widrow and Marcian Hoff were significant advances [154], [155]. The Widrow-Hoff rule is very general and still finds frequent application today. Rosenblatt's publication of "Principles of Neurodynamics"[137] in 1962 energized the field by presenting a well developed theory to accompany the perceptron. Still, there were troubles. Perceptron net-

works could not always solve the problems which they were presented. Even some very simple problems could not be solved, no matter how large the network was. The bubble burst in 1969 when Marvin Minski and Seymour Papert published "Perceptrons"[117] in which they analyzed the perceptron and showed that a single layer of perceptrons was incapable of solving certain problems. A shadow of doubt was also cast upon the multi-layer perceptron networks. Little could be done with them at the time since no learning rule to adjust the network connections was available. The following is taken from "Perceptrons", pp. 231-232.

> The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgment that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting "learning theorem" for the multilayered machine will be found.

In the wake of this prediction regarding multi-layer perceptrons, the entire field of neural network study practically vanished over night! The development of that interesting learning theorem was only five years away but it would languish in obscurity for another twelve years beyond that.

During the seventies scattered researchers continued their work on various types of neural networks. Kunihiko Fukushima [47] developed a class of network architectures known as neocognitrons for biologically motivated visual pattern recognition. Tuevo Kohonen[70] and James Anderson[13] were working independently on associative memo-

ries which are more behaviorally based and less structurally based on brain-like activity. Stephen Grossberg[60] developed network architectures and theories that eventually led to his Adaptive Resonance Theory. Although Minsky and Papert's "interesting learning theorem," the solution to the problem of training networks consisting of multiple layers of perceptron, was developed in 1974 by Paul Werbos [193], it went completely unnoticed by neural network researchers until much later. Another development that achieved little notoriety at the time was the 1972 doctoral dissertation by James Albus in which he described his idea for the Cerebellar Model Articulation Controller (CMAC)[2]. It is interesting to note that the mathematical model of cerebellar function developed by Albus in 1971 [1] was also developed independently by David Marr [90] in Great Britain in 1969. Much more will be said about the CMAC since its analysis and improvement are the central topics of this work.

The learning rule developed by Werbos and now known as backpropagation first attained widespread recognition by neural network researchers in 1986 when David Rumelhart, Geoffrey Hinton and Ronald Williams[139] presented their independent development of error backpropagation for the training of multi-layer perceptron networks. It was soon discovered that D. B. Parker had also independently developed the same algorithm in 1982 and called it Learning Logic. The backpropagation algorithm caused a tremendous resurgence of interest in neural networks (particularly in multi-layer perceptron networks).

The CMAC algorithm also began receiving some new attention in the 1980's. Ersu et. al. [36] began working with the CMAC algorithm in 1981. They called their algorithm the Associative Memory System (AMS) and applied it primarily to simulated systems or

to systems with slow dynamics (sampling rates on the order of 1 second). By 1986 W. Thomas Miller [99], at the University of New Hampshire, had refined the Albus CMAC to the point where it could be applied to the real-time control of complex systems with faster dynamics (15 states, sampling rates on the order of 10 milliseconds). Dedicated CMAC hardware was developed at the University of New Hampshire in 1990 reducing CMAC response time to 1 millisecond or less.

It was in the late 1980's that a detailed analysis of the CMAC algorithm began. Ellison, in 1988, first analyzed the convergence properties of the Albus CMAC. Subsequently, Militzer and Parks [94][128], Kraft and Campagna [74], Wong and Sideris [158], and Brown [19] have all studied the learning convergence of the CMAC neural network. Carter et. al. [28] has studied the fault tolerance of the CMAC.

1.3 Some Examples of CMAC Applications

The Robotics Laboratory at the University of New Hampshire has been applying the CMAC network to the solution of a wide assortment of problems since the mid 1980's. Applications of the CMAC neural network include the following: "Application of a Simple cerebellar model to geologic surface mapping"[61] (1991), "Deconvolution and Nonlinear inverse filtering using a neural network"[56] (1989), "Deconvolution using a CMAC neural network"[57] (1988), "Application of a general learning algorithm to the control of robotic manipulators"[102] (1987), "Real time application of neural networks for sensor-based control of robots with vision"[104] (1989), "Pattern Recognition using a CMAC Based Learning System"[63] (1988), "Practical demonstration of a learning control system for a five axis industrial robot"[64] (1988), "Real time dynamic control of an

industrial manipulator using a neural network based learning controller"[105] (1990), "Bipedal Gait Adaptation for Walking with Dynamic Balance"[96] (1991), "Real-Time Neural network Control of a Biped Walking Robot"[95] (1994), "Rapid Learning Using CMAC Neural Networks: Real Time Control of an Unstable System"[97] (1990), "On-Line Hand-Printed Character Recognition Using CMAC Neural Networks"[115] (1994), "A CMAC-based cursive handwriting recognizer for the Windows for Pen Computing operating environment"[14] (1994), Real-Time Control of a Robotic Pole Balancing System Using Complementary Neural Network and Optimal Techniques"[122] (1994), "Approximate nonlinear optimal control using CMAC neural networks"[12] (1994), "Control of pH Using a Self-Organizing Control Concept with Associative Memories"[38] (1983), "Shape recognition using a CMAC based learning system"[55] (1987), "Application of Associative memory Neural Networks to the Control of a Switched Reluctance Motor"[135] (1993), "Dynamic Control of a Parallel Link Manipulator using a CMAC Neural network"[54] (1993), and "Analyzing Biological Signals with CMAC, A Neural Network"[156] (1991).

1.4 A Brief History of Adaptive Controls

The field of adaptive controls evolved essentially in parallel with, yet isolated from, the field of neural networks. Where neural network research attempted to produce some rudimentary brain-like behavior from their algorithms, the adaptive control researchers were developing solutions to problems by extending the existing linear system theory with adjustable parameters and supporting this extension with mathematical rigor.

The process of adaptive control can typically be divided into two steps: system identification and system control. Adaptive control techniques are applied when the plant

to be controlled has unknown parameters but an assumed linear functional form. System identification is the process of converting plant input-output measurements into estimates of the linear function parameters. These new plant parameter estimates are then used to modify parameters in the system controller with the goals of stable control and good performance relative to some performance measure.

It is also possible to combine the two steps of identification and control so that they occur simultaneously in a process called simultaneous identification and control. Under this approach the controller parameters are adjusted directly based upon system input-output measurements. This scheme is also referred to as direct control for this reason. Control with separate identification and control is also referred to as indirect control. For linear systems this theory is well developed and includes detailed stability analyses. Two different approaches to this adaptive control design are the Model Reference Adaptive Control [194], [186] (MRAC) strategy and the Self Tuning Regulator [175], [165] (STR) strategy.

The goal of stable control with good performance is very elusive. Theoretical guarantees of system stability and performance are difficult to achieve even under very restrictive assumptions. MRAC systems are designed from the standpoint of guaranteed stability while STR systems have no guaranteed stability results. Both have found wide application and both have advantages and disadvantages governing where they may best be applied.

There are typically two kinds of control problems to be solved. The category into which a particular control problems falls depends primarily upon the control objective. Essentially, control is about keeping the plant outputs Y_p within some prescribed limits. When these limits are defined in terms of some "desired" plant behavior Y_d, the problem

is of the model reference type. The goal is to keep the error $e = Y_p - Y_d$ within the prescribed limits. The choice of mathematical form for Y_d is important for tractability and Y_d is usually implemented as the output of a reference model -- hence the name of this control strategy. If Y_d is a constant, the control problem is referred to as "regulation" and if Y_d varies with time, the control problem is referred to as "tracking." Both direct and indirect implementations of MRAC are possible.

It can be argued that the first mainstream adaptive control paradigm was the "M.I.T. Rule" developed by Whitaker et al [194] of the Massachusetts Institute of Technology in 1961. The M.I.T. Rule was a heuristic method for designing a model reference controller where the parameters of the controller were adjusted to minimize an error function using a gradient approach. The M.I.T Rule was difficult to analyze and Parks and others soon discovered that it could lead to instabilities even for simple systems. The generally accepted solution to this problem was to redesign the model reference adaptive controller using Lyapunov's direct method [186], [180] so that system stability was guaranteed from the outset.

The self-tuning regulator was originally proposed by Kalman [175] in 1958 for the stochastic minimum variance control problem. The STR has been studied extensively and many variations exist [165], [166]. STRs find application in noisy systems and are divided into two subcategories, explicit estimation and implicit estimation, corresponding to indirect and direct MRAC respectively . Explicit STRs consist of an explicit estimation of the process to be controlled followed by a tuning of the regulator parameters; implicit STRs are based on an implicit estimation of the process and a direct update of the regulator parameters.

1.5 Prior Work in CMAC Learning Convergence

There are several groups of researchers who have published or are working on proofs of CMAC learning convergence. All of these proofs are for the open-loop learning case. Openloop learning is defined to be the case where a CMAC neural network is placed in parallel with a system to be modeled. The CMAC is trained to duplicate the input-output relationship of the system being learned. Figure 1.1 shows a typical block diagram.

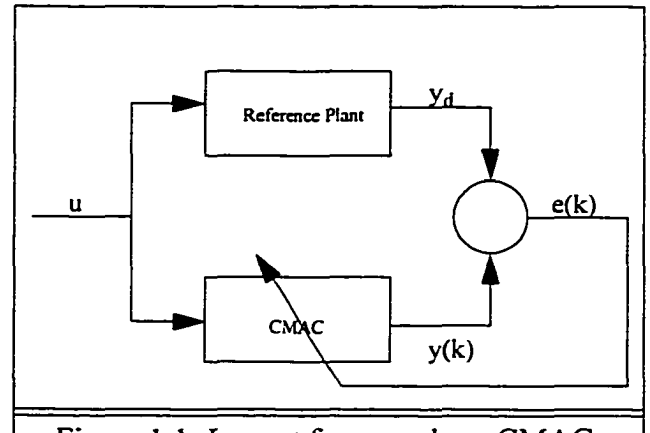


Figure 1.1: Layout for open-loop CMAC training.The CMAC is trained to duplicate the input-output behavior of the Reference Plant.

In 1988 Ellison published "On the Convergence of the Albus Perceptron" [34] where it was proved using a rank argument on the system of linear equations solved by the CMAC that, for the 1-dimensional case, the optimum weight vector for which the output error was zero always existed. Since the 1-dimensional case has fewer equations than unknowns, there are infinitely many weight vectors for which the output error is zero.

Parks and Militzer published their CMAC convergence proof in 1989 [128]. The analysis was based on a geometrical interpretation of the CMAC algorithm as a projection operation in the weight space. When convergence to a point in the weight space was indi-

cated (i.e. a unique weight vector exists), a Lyapunov function was constructed to prove convergence. When the convergence was to a limit cycle, an eigenvalue analysis was used to determine that all the eigenvalues were either on or inside the unit circle. In another paper, also published in 1989 (a similar paper, "A Comparison of Five Algorithms for the Training of CMAC Memories for Learning Control Systems", was published in Automatica in 1992), Parks and Militzer examined the convergence properties of five different CMAC update laws [130]. One of these update laws was the standard CMAC law for which they had already proved convergence. Convergence proofs were not developed for the other training rules.

In 1991 Ellison published a paper "On the Convergence of the Multidimensional Albus Perceptron" [35]. 1992 saw more proof papers on the convergence of CMAC. First, in January, Wong and Sideras [158] equated the CMAC algorithm with iterative Gauss-Seidel solution for a set of linear equations and claimed that CMAC always converged to arbitrarily small output error. They presented a detailed proof for a 1-dimensional CMAC (maps $\Re \rightarrow \Re$). They also claimed that the proof could be extended to multidimensional CMACs (maps $\Re^n \rightarrow \Re^m$) but Brown and Harris [21] have demonstrated that this extension does not always exist in the multidimensional case.

In 1993, Campagna and Kraft first used a Lyapunov function of the weight error between the CMAC being trained and a fully trained CMAC representing the desired function to show that learning always converges for values of the learning rate parameter between 0 and 2 as long as it could be postulated that a target set of weights existed. This analysis was an extension of some of Parks and Militzer's analysis in that the learning rate parameter was allowed to vary as opposed to being fixed at the value 1. Brown, in his

work on Neurofuzzy Adaptive Modeling and Control [19], developed several theorems on CMAC's ability to exactly model certain classes of functions.

Chapter 2

The CMAC Neural Network

2.1 CMAC Description and History

The CMAC algorithm, developed by James Albus in the early 1970's as a model of human cerebellar functioning suitable for function approximation and manipulator control, is a mapping from a set of possible inputs to a collection of weights. The CMAC is based on the perceptron developed by Rosenblatt and it has the following properties:

- Each of the inputs maps to exactly C of the weights whose values are summed together to produce the CMAC output. This number C is often called the generalization size.
- Inputs that are "similar" or "close" to each other in the input space will map to many, but not all, of the same weights thus producing outputs that are also "similar." This property is known as generalization and makes it possible for the CMAC to produce an appropriate (or approximately so) response to an input that has not been presented before if that input is sufficiently similar to data the CMAC has already trained on.
- If the inputs are sufficiently "distant" from each other in the input space, the outputs will be independent.

Figure 2.1 shows the high-level representation of the CMAC algorithm illustrating the mapping of sets from the input space to sets of real memory location in the space of

computer memory. An input vector is a collection of N measurements representing sys-

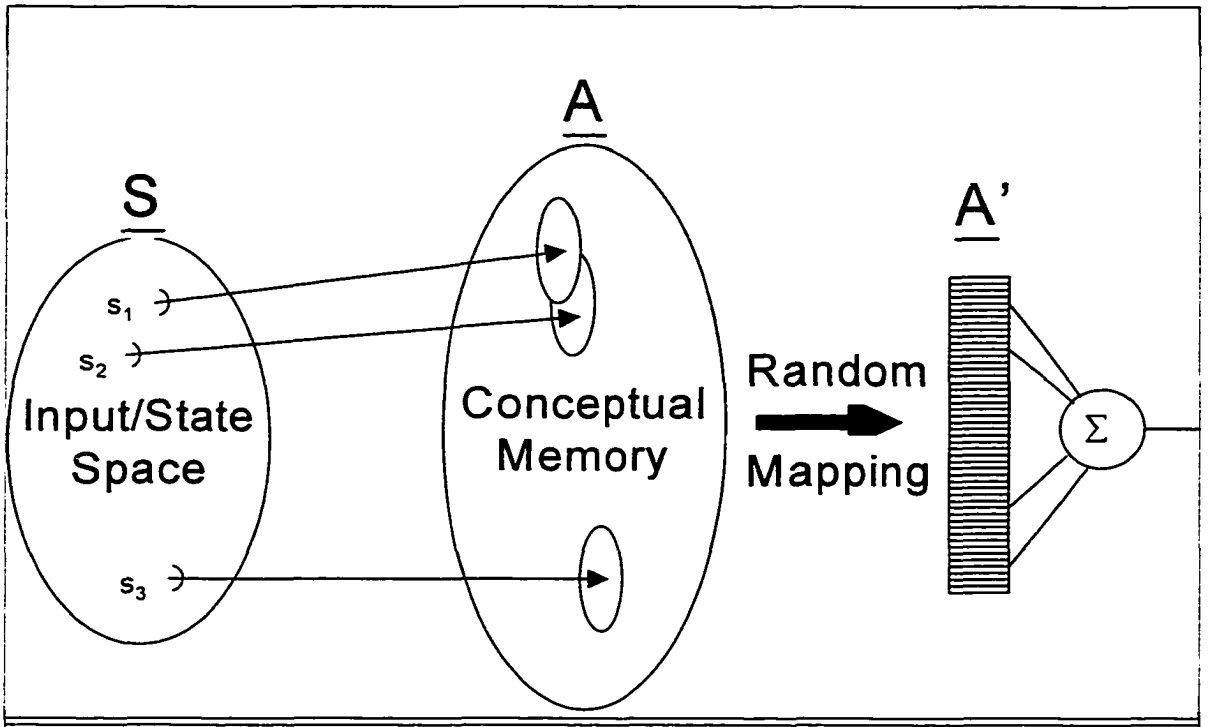


Figure 2.1: High-level, set-oriented representation of the CMAC.

tem inputs, systems states, and/or desired system responses. Each input maps via the CMAC algorithm to a set of conceptual memory locations in the space labelled A in figure 2.1. The number of elements in the input space grows very rapidly with the number of inputs and particularly with the number of discrete values each of these inputs may take on. Imagine, for instance, a CMAC with four inputs where each input is sampled analog data with 10-bit resolution. The resulting input space has 2^{40} or over 1 trillion possible input vectors. The corresponding A memory must be at least as large. Since most systems only visit a small fraction of their total possible input space, a technique called hashing is used to map the A memory into a more practically sized A' memory. While hashing leads to a tremendous reduction in the memory requirement of the CMAC, this benefit is not without an attached cost. Hashing makes it possible for widely separated vectors in the

input space to be mapped to one or more of the same weights thereby violating the third of the above CMAC properties and resulting in a condition known as a collision or hashing collision. This problem can be mediated to some extent by utilizing certain collision avoidance enhancements to the algorithm.

Figure 2.2 is a diagram of a two-input Albus-type CMAC as implemented by the basic UNH CMAC algorithm. Each variable in the input vector $\underline{S}$ is processed by a col-

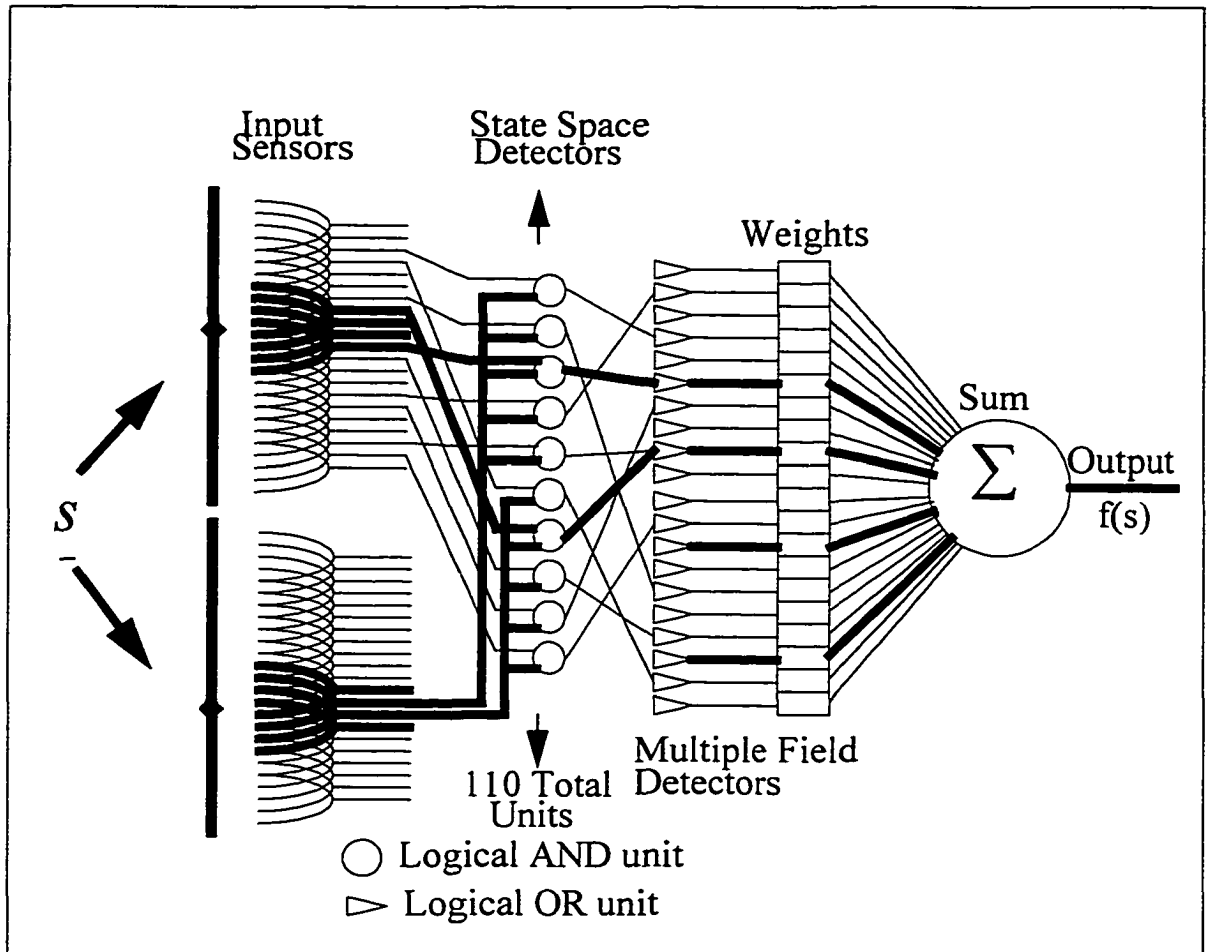


Figure 2.2: A simple example of a CMAC neural network with two inputs and one output. The generalization parameter, C, is set to 4 and only a partial set of state space detectors is shown.

lection of input sensors with overlapping regions of sensitivity or receptive fields. In this case, each input sensor produces a binary output which is ON if the input falls within its

receptive field and is OFF otherwise. Because of this, the Albus CMAC is often referred to as the Binary CMAC. The width of the receptive field of each sensor is what leads to input generalization. The offset of the adjacent receptive fields is a measure of the imposed quantization of the input space. In a sampled data system the input space is already broken up by the sampling process into a number of discrete values determined by the number of bits used to represent the data. The CMAC quantization parameter determines the distance, measured in terms of some number of these discrete values, that an input must change by in order to activate a new receptive field.

A lower resolution view of an input variable is often quite sufficient for learning applications and sometimes masking small variations even helps to prevent or reduce the learning of system noise. Each input variable excites exactly C input sensors where C is the ratio of generalization width to quantization width. For the rest of this work it will be assumed that the parameter C and the generalization width have the same value. This, of course, implies a quantization parameter value of unity.

The binary outputs of the receptive fields are combined in a series of logical AND units called state-space detectors. Each state-space detector receives one input from the receptive field sensors for each of the input variables. Only the state-space detectors having all inputs in the logical ON state produce an output in the logical ON state. This effectively creates a new sensor with a multidimensional receptive field. For the two-dimensional case, the receptive field of a state-space detector is the interior of a square in the input space. In general the state-space detector receptive field is the interior of a hypercube in the input space. The state-space detectors are selectors for memory locations in the A memory.

If the receptive field sensors were fully connected to the state-space detectors then C^N memory locations in the A memory would be selected. To avoid having an unmanageable number of state-space detectors, the input sensors are connected in a sparse and regular way such that exactly C state-space detectors are ON for any input vector. An effective way to visualize the input space coverage provided by this set of state-space detectors is to divide them into C classes such that the receptive fields for the detectors in each class completely cover the input space without overlap. These hyper-planes are then stacked with their origins displaced from one another in a predefined way. Each vector in the input space excites exactly C state-space detectors. The Albus receptive field placement results in an ordered distribution of receptive fields along the main hyper-diagonal and parallel minor diagonals of the input space. In other words, the displacement of the receptive field hyper-planes is positive one unit along each axis of the input space.

Although the Albus receptive field placement is easy to calculate and it does produce even, regular coverage in the input space, it results in highly nonuniform sampling in the weight space. The larger the value of C and the higher the input space dimension, the greater the non-uniformity becomes. This phenomenon along with alternative receptive field center distribution schemes has been explored in detail [9], [66], [129]. The effect of this nonuniform sampling is to produce excellent generalization along certain trajectories in the input space and poor generalization along other trajectories. An [9], in his 1991 doctoral dissertation, developed a heuristic receptive field placement that produced a much more uniform distribution in the weight space. The solution explored by the above researchers is to arrange the receptive field centers in a manner analogous to sphere packing. This new mapping from input sensors to state-space detectors still results in uniform

sampling in the input space and approximately uniform sampling in the weight space. Yan-Ping Jia [66] in her masters thesis explored CMAC weight allocation using the mathematically rigorous results obtained from the exploration of the sphere packing problem. To date, results are only available for low dimension problems.

The CMAC was more widely used in control systems beginning in the early 1980's with Ersu et. al. who applied it primarily to simulated systems or to systems with slow dynamics (sampling rates on the order of 1 second). They called their algorithm the Associative Memory System or AMS. By the mid 1980's, Miller had refined the CMAC algorithm to the point where it could be applied to the real time control of complex systems with faster dynamics (15 states, sampling rates on the order of 10 milliseconds). In 1990 hardware was developed that reduced CMAC response time to 1 millisecond or less depending upon the generalization size. Reay et. al.[135] have also been using the CMAC as part of their control system research for the control of switched reluctance motors.

There has also been much theoretical interest in the CMAC. Militzer and Parks[94], Kraft and Campagna[74], Wong and Sideris[158], and Brown[19] have all studied the learning convergence of the CMAC neural network. Carter et. al.[28] has studied the fault tolerance of the CMAC.

Much can now be said about the properties of the CMAC and some of the important ones are summarized below:

- Since the CMAC is a local learning algorithm, only a small number of network weights need to be accessed on any pass through the network. This makes the CMAC especially well suited to real-time applications and allows for good performance even on relatively inexpensive general purpose hardware.

- The theoretical analysis of CMAC shows that learning convergence can often be guaranteed in open-loop systems. The empirical evidence shows that it is usually well behaved in closed-loop systems as well.
- In conjunction with even a poorly tuned controller, the CMAC generalization property often enables the network to essentially guide its own learning by progressively assuming control of the plant. This makes the CMAC an excellent tool for system performance enhancement since it can compensate for small system nonlinearities which have been ignored or linearized for the design of a traditional linear controller.
- Although trained information is distributed across a large number of network weights, the CMAC is still susceptible to the phenomenon of single weight dominance where one or two weights are the primary contributors to the network response. One consequence of this is the occasional uncontrolled growth of a single weight leading to dynamic range problems, particularly with integer weights. This also reduces the fault tolerance of the network considerably.
- Partial derivative information derived from the CMAC output is extremely unreliable, particularly in regions where the training exemplar density is low. Local generalization effects often lead to large magnitude errors and even sign errors in calculated partial derivatives.

2.2 CMAC Parameters

A detailed explanation of the parameters is left to the user's guide in the appendix. These parameters are for the integer CMAC typically used at the University of New Hampshire.

2.2.1 Quant

The quantization parameter **quant** determines the amount of sampling performed

on the CMAC input variables. If the value of **quant** is 1, then all integer points in the input space are available for training. If the value of **quant** is set equal to 2, then the sampling is for every other sample in the input space. This parameter can be used, for example, to ignore the noisy least significant bits of data provided by some analog-to-digital converter.

2.2.2 Num_state

The parameter **num_state** tells the CMAC how many input states there are. For the 2-dimensional problems being studied here, **num_state** is always 2.

2.2.3 Num_resp

The parameter **num_resp** is the number of responses the CMAC is to provide for each input vector presented. An independent set of weights is allocated for each response.

2.2.4 Num_cell

The parameter **num_cell** is the number of CMAC weights across which the input information will be spread during training. In the standard CMAC, it is also the number of weights which are summed together to produce an output during the remember phase. This parameter is also known as the generalization parameter.

2.2.5 Mem_size

The parameter **mem_size** represents the number of integer memory words allocated to store the information trained into the CMAC. Mem_size locations are allocated for each of the desired outputs plus mem_size locations for collision avoidance tracking.

2.2.6 Collision_avoidance

When collision avoidance mode is enabled, learning performance is relatively independent of the amount of memory allocated as long as there is enough. Performance degradation is fairly abrupt when collision avoidance is enabled whereas it is gradual

when there is no collision avoidance. A CMAC without collision avoidance enabled requires much more memory than does a CMAC with collision avoidance to achieve similar learning performance. The parameter **collision_avoidance** takes on values of 1 for avoidance disabled and 0 for avoidance enabled. Collision avoidance mode does not eliminate the possibility of a learning collision; it merely makes such an event much less likely.

2.2.7 CMAC_beta

The parameter **cmac_beta**, traditionally knows as beta, determines the weighting of new information as it is combined with existing information stored in the CMAC. From a theoretical standpoint, the value of **cmac_beta** can vary over the continuum from 0 to 2 with typical use restricted to the range from 0 to 1. For the UNH CMAC, **cmac_beta** takes on positive integer values beginning at 0. Since the UNH CMAC is an all-integer network, integer values of **cmac_beta** are converted to multiplicative scaling factors of the form $1/2^{\text{cmac\_beta}}$ which can be implemented very efficiently in the computer as binary shifts.

In this thesis we will attempt to motivate "weight smoothing" during learning as a way to improve CMAC generalization, fault tolerance, and the quality of partial derivatives calculated from partially trained CMAC function approximations. The methodology will be to represent the CMAC as a collection of linear equations in the network weights. We will treat this CMAC representation as a constrained optimization problem and demonstrate the potential for improved performance available from this batch mode training. We will then take the batch mode optimization result and develop an iterative form to more closely represent the typical CMAC and comment on the results.

2.3 The Matlab CMAC Toolbox

The basis for much of the research in this dissertation is the publicly available UNH CMAC C code. Since it was known that visualization would play a key role in this research, one of the first tasks was the porting of the UNH CMAC code to the Matlab environment. The resulting MAT_CMAC toolbox has already been used as a research tool in other thesis work [46]. The process of creating the MAT_CMAC toolbox consisted of writing a Matlab mexfunction wrapper in C for each of the externally available functions in the UNH CMAC code. In addition to the straight port of the CMAC code, numerous extensions were added in support of this research. Table 2.1 is a listing of the

Table 2.1 MAT_CMAC neural network toolbox

adjust	update CMAC weights with a correction you determine
aloccmac	create a new CMAC
clrwgts	set all CMAC weights to zero
delcmac	delete a CMAC from the matlab environment
learn	update CMAC weights with standard algorithm
memusage	estimate memory usage by finding all nonzero weights
*reg_cmac	register a new CMAC with the CMAC manager
remember	get a response(s) from a CMAC
rstrcmac	create a new CMAC and load it from a file
savecmac	save a CMAC to a file
mapinput	get weights and receptive field shapes corresponding to an input
*nh_coord	get coordinates of receptive field corners activated by a particular input
setrfdis	set receptive field displacement vector for a CMAC
*getrfdis	get receptive field displacement vector and size from a CMAC
setrfmag	set receptive field magnitude table for a CMAC
*getrfmag	get receptive field magnitude table and size from a CMAC
*set_gen	set CMAC generalization parameter (use with caution)

Table 2.1 MAT_CMAC neural network toolbox

*get_gen	get CMAC generalization parameter
*comcoord	find coordinates common to two lists of coordinates
*coomatch	generate a matrix indexed by state representing number of matches between two lists of coordinates
*coomat2	faster version of coomatch for most cases
*coomat2n	normalized to only count matches once
*set_sgen	set CMAC super generalization parameter
*get_sgen	get CMAC super generalization parameter

MAT_CMAC toolbox functions. An asterisk beside the function name indicates that it is an extension developed as part of this research. Compiled MAT_CMAC toolboxes exist for both Matlab 4.2 and Matlab 5.1 on the PC. It has also been successfully compiled under UNIX on an Indigo workstation. The PC compilations were performed using Microsoft Visual C++ ver 5.0.

Chapter 3

CMAC Stability

In this chapter the stability and convergence properties of CMAC networks are analyzed. This dissertation deals only with the open-loop case. The ability of CMAC to learn functions is studied. The class of linear and nonlinear systems which can be adequately represented by a CMAC neural network is considered for this effort. The novel aspect of this approach to a convergence proof for the CMAC algorithm is that convergence is shown not to the original system, but rather to its CMAC equivalent. Figure 3.1

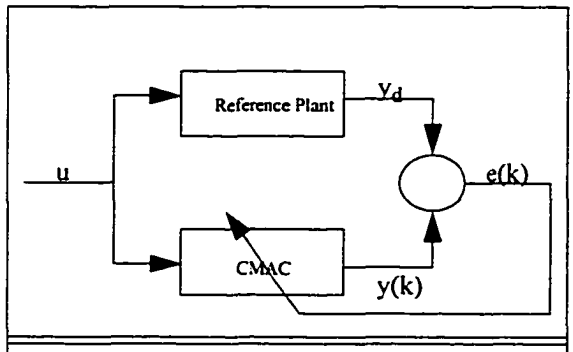


Figure 3.1: Layout for open-loop CMAC training.

shows the block diagram for open-loop CMAC training in the single-input, single-output case. The assumptions are:

- The reference plant is represented with arbitrary accuracy, over some region of interest, as a completely trained CMAC neural network.

- The reference plant CMAC and the CMAC being trained have the same size and form i.e. the same input, u, will select the same weights from the respective weight vectors to produce the output.

3.1 Proof for Open-Loop Learning in the Scalar Case

Define Y_d as the vector of all possible outputs from the reference plant CMAC where $\underline{S}$ is the selection matrix and $\underline{w}_d$ is the constant vector of weights.

$$\underline{Y}_d = \underline{S}\underline{w}_d \qquad \text{Eqn 3.1}$$

The selection matrix $\underline{S}$ is determined by the mapping from the input variable, u, to the weight space as defined by the CMAC algorithm. The input, u, is quantized into m distinct levels resulting in the selection matrix, $\underline{S} \in \{0, 1\}^{m \times n}$, where n is the number of weights in the weight vector $\underline{w}_d$. This function causes a particular input u to map to a unique row of $\underline{S}$. This relationship is denoted by:

$$\underline{s}_{i(k)} = ith \text{ row of } \underline{S} \text{ at time k}$$

where the subscript for a row of $\underline{S}$ is:

$$i(k) = f(u(k)) \qquad \text{Eqn 3.2}$$

Each row, $\underline{s}_{i(k)}$, of $\underline{S}$ has exactly C (generalization size) ones with all other elements in the row being zero. The time index k in the subscript denotes that the active row of the selection matrix can vary from time step to time step as would be expected with a time varying input.

The ith output, $y_{di(k)}$, is defined as follows:

$$y_{di(k)} = \underline{s}_{i(k)}\underline{w}_d. \qquad \text{Eqn 3.3}$$

$$\underline{w}_d \in \Re^{n \times 1}, fixed \qquad \underline{s}_{i(k)} \in \{0, 1\}^{1 \times n}$$

The defining equations for the output of the CMAC being trained are given below. As a result of the second assumption above, the selection matrix, $\underline{S}$, is the same for both CMACs.

$$\underline{Y} = \underline{S}\underline{w}(k) \qquad \text{Eqn 3.4}$$

$$y_{i(k)} = \underline{s}_{i(k)}\underline{w}(k) \qquad \text{Eqn 3.5}$$

$$\underline{w} \in \Re^{n \times 1} \qquad \underline{s}_{i(k)} \in \{0, 1\}^{1 \times n}$$

Define $e(k)$ to be the output error at the kth time step. This error is directly a result of the weight error between the reference CMAC and the training CMAC, denoted as $\Delta\underline{w}(k)$.

$$e(k) = \underline{s}_{i(k)}(\underline{w}_d - \underline{w}(k)) = \underline{s}_{i(k)}\Delta\underline{w}(k) \qquad \text{Eqn 3.6}$$

Select the following Lyapunov function:

$$V(k) = \Delta\underline{w}(k)^T\Delta\underline{w}(k) \qquad \text{Eqn 3.7}$$

The goal is to show:

$$V(k+1) \leq V(k) \quad \forall k \qquad \text{Eqn 3.8}$$

Choose the standard CMAC update law and write it as follows

$$\Delta\underline{w}(k+1) - \Delta\underline{w}(k) = -\underline{s}_{i(k)}^T\left(\frac{\beta e(k)}{C}\right) = -\underline{s}_{i(k)}^T\left(\frac{\beta \underline{s}_{i(k)}\Delta\underline{w}(k)}{C}\right) \qquad \text{Eqn 3.9}$$

where β is the learning rate.

Substituting into $V(k+1)$ gives:

$$V(k+1) = V(k) + (\beta^2 - 2\beta)\frac{e^2(k)}{C} \qquad \text{Eqn 3.10}$$

Since the term $\beta^2 - 2\beta$ is strictly less than zero for all β such that $0 < \beta < 2$ and since e^2

and C are both strictly greater than zero, then $V(k+1) \leq V(k)$ and the learning does not diverge by the second method of Lyapunov. Learning continues until there is zero output error. Under conditions of persistant excitation, where all desired input states are repeatedly visited, the condition of zero output error for all inputs equates to zero weight error since the output error is directly related to error in some of the weights by Eqn 3.6.

3.2 Extension to the Multiple-Input, Single-Output Case

The above results are now extended to the case where the input to the CMAC is composed of multiple signals.

In this case,

$$\underline{u}(k) = [u_1(k) u_2(k) \ldots u_N(k)]^T \qquad \text{Eqn 3.11}$$

which leads to Eqn 3.1 where

$$\underline{S} \in \{0, 1\}^{\left(\sum_{i=1}^{N} m_i\right) \times n}$$

and m_i is the quantization level of the ith component of the input vector.

From this point the proof is identical to the proof for the single-input, single-output case. While there is good agreement that the single-input, single-output CMAC will always converge ([19], [158]), the conditions under which the multidimensional CMAC error will converge to zero are less well understood. This proof shows that the excited weights in the trainable network will converge to the corresponding target weights.

3.3 Extension to the Multiple-Input, Multiple-Output Case

The extension to the multiple-input, multiple-output case can be accomplished by assigning a multiple-input, single output CMAC to each of the outputs being learned.

$$[\underline{Y}_1 \underline{Y}_2 \cdots \underline{Y}_M] = \underline{S}[\underline{w}_1 \underline{w}_2 \cdots \underline{w}_M] \qquad \text{Eqn 3.12}$$

This representation is consistent with the practice of vector weights thereby enforcing a requirement that all CMACs use the same selection matrix. The following Lyapunov function is chosen:

$$V(k) = \sum_{j=1}^{M} \Delta \underline{w}_j(k)^T \Delta \underline{w}_j(k) = \sum_{j=1}^{M} V_j(k) \qquad \text{Eqn 3.13}$$

Since the individual $V_j(k)$ have been shown to be Lyapunov functions, the sum must be also. This result indicates that the multiple-input, multiple-output CMAC will converge provided that the individual multiple- (or single)-input, single-output CMACs converge.

3.4 Discussion

This dissertation presents a Lyapunov-based proof for the convergence of open-loop learning for CMAC. The only assumptions are that (1) the function being learned can be adequately represented by a fully trained target CMAC and (2) both CMACs are the same size so that their weight selection properties are the same.

The issue of the existence of a fully trained target CMAC is not addressed in this paper. Brown [19] examines some of these issues in his work. Further, this proof in no way guarantees that the weights being updated will eventually match the weights of the fixed network. It says only that the Lyapunov function will decrease for as long as there is a nonzero output error. At the point where the output error is zero for all training inputs, the CMAC weights will have converged to a solution, but the possible equality in Eqn 3.8 allows for the existence of a nonzero weight error. Computer simulations have shown that the Lyapunov function decreases monotonically until learning is complete even though the system output error does not decrease monotonically.

In order to make a statement about global asymptotic convergence in the weight space, a requirement similar to observability in linear systems needs to be imposed. All weights must appear in the output error in a linearly independent manner. This requires persistent excitation of all the weights by the input. Asymptotic convergence of all the weights would be guaranteed under these conditions. Efforts in this dissertation have centered on forcing the CMAC weights to converge to a particular solution instead of allowing it to select from the large family of acceptable weight solutions as is the case with the present algorithm.

Chapter 4

CMAC Weight Smoothing

4.1 The CMAC as a Collection of Linear Equations

Begin by defining $\underline{Y}$ as the vector of all possible outputs from the CMAC where $\underline{S}$ is called the selection matrix and $\underline{w}$ is the vector of weights.

$$\underline{Y} = \underline{S}\underline{w} \qquad \text{Eqn 4.1}$$

We are interested in the case of partial network training so there will be fewer training points than the total number of possible points in the quantized input space. In fact, for the case of all dimensions higher than one, training at all or almost all the possible inputs in the quantized space results in a selection matrix which is rank deficient. The allowable percentage of trainable input points before rank deficiency occurs also varies with the receptive field center placement strategy.

The selection matrix $\underline{S}$ is determined by the mapping from the input variable, u, to the weight space as defined by the CMAC algorithm. For the matrix analysis, the final step of hashing the weight memory addresses for improved storage has been omitted. For the Albus CMAC, the input, u, is quantized into m distinct levels resulting in the selection matrix, $\underline{S} \in \{0, 1\}^{r \times n}$, where n is the number of weights in the weight vector $\underline{w}$ and $r < m$ is the number of states in the quantized space which are actually used to train the CMAC. Details of this map may be found in [2], [107]. Each quantized input $u(k)$ maps to

a unique row of $\underline{S}$. This relationship is denoted by:

$$\underline{s}_{i(k)} = ith \text{ row of } \underline{S} \text{ at time k}$$

where the subscript for a row of $\underline{S}$ is:

$$i(k) = f(u(k)) \qquad \text{Eqn 4.2}$$

For the Albus CMAC each row, $\underline{s}_{i(k)}$, of $\underline{S}$ has exactly C (generalization size) ones with all other elements in the row being zero. The time index k in the subscript denotes that the active row of the selection matrix can vary from time step to time step as would be expected with a time varying input.

The ith output, $y_{i(k)}$, is a scalar in this discussion, although a vector formulation can be developed, and is defined as follows:

$$y_{i(k)} = \underline{s}_{i(k)}\underline{w}. \qquad \text{Eqn 4.3}$$

$$\underline{w} \in \Re^{n \times 1} \qquad \underline{s}_{i(k)} \in \{0, 1\}^{1 \times n}$$

A cost function can be defined as follows for the constrained optimization problem with Z representing the training sample vector:

$$J = \frac{1}{2}\underline{w}^t\underline{\underline{Q}}\underline{w} + \underline{\lambda}^t(\underline{\underline{S}}\underline{w} - \underline{Z}) \qquad \text{Eqn 4.4}$$

More will be said about the $\underline{\underline{Q}}$ matrix shortly. It is this matrix that determines the extra constraints the network weights must satisfy in addition to solving the original set of linear equations.

Finding the minimum of the cost equation in Eqn 4.4 selects the weight which best meets the constraints specified. For this research, the weight vector having minimum difference between adjacent values and also having small magnitudes is the vector selected

by this process. Taking the partial derivative of the cost function with respect to the weights,

$$\nabla_w J = \underline{Q}\underline{w}^t + \underline{S}^t\underline{\lambda} \qquad \text{Eqn 4.5}$$

and invoking the first order necessary condition ($\nabla_w J = 0$) for the minimization of J, $\nabla_w J = \underline{0}$ yields

$$\underline{w} = \underline{Q}^{-1}\underline{S}^t(\underline{S}\underline{Q}^{-1}\underline{S}^t)^{-1}\underline{Z} \qquad \text{Eqn 4.6}$$

The solution represented by Eqn 4.6 is a batch mode weight update law in that it requires all available training data to be present. It will be optimum with regard to whatever penalty matrix $\underline{Q}$ is chosen. In general, the $\underline{Q}$ matrix must be symmetric and positive definite. For example, the minimum length $\underline{Q}$ matrix is the identity matrix while the minimum difference $\underline{Q}$ matrix takes the following form for a case with 4 weights and extends along the diagonal for cases with a greater number of weights:

$$\underline{Q} = \alpha\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} + \beta\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{with } (\alpha \geq 0)\text{and}(\beta > 0) \qquad \text{Eqn 4.7}$$

The portion of the $\underline{Q}$ matrix pre-multiplied by α represents a penalty on the difference between adjacent weights. This difference penalty is used to enforce the desired amount of smoothness in the weights. The portion of the $\underline{Q}$ matrix pre-multiplied by β represents a penalty on the magnitude of the weights and is the minimum length penalty. A small amount of magnitude control is usually applied so that $\underline{Q}$ is non-singular. This minimum weight magnitude portion of the $\underline{Q}$ matrix also serves to control the problem of certain CMAC weights creeping off to infinity as the result of repeated, uncompensated updates during learning, thereby stabilizing the CMAC weight dynamic range.

4.2 Some interpretations

Many optimizations are possible. The optimization of quadratic costs is the focus of this work. Several important parameters are seen to influence the nature of the optimization. These include: the definition of the term being optimized, the choice of penalty matrix $\underline{Q}$, and, for these CMAC optimizations, influences on the weight selection matrix, $\underline{S}$, such as the receptive field placement strategy and whether the receptive fields are tapered. Several optimizations are described below along with the traditional CMAC update law. The goal of this research is to extract information from formulating the CMAC as an optimization problem and then apply it to the CMAC learning algorithm.

4.2.1 Matrix Formulation of the Albus CMAC Weight Update Law

The Albus CMAC weight update law is a local update law. Network weights are updated according to the following rule:

$$e = d - \frac{(s_{i(k)} w_o)}{C} \qquad \text{Eqn 4.8}$$

$$w = w_o + s_{i(k)}{}^t \beta e \qquad \text{Eqn 4.9}$$

where w_o is the weight vector before the new data point appeared, d is the new data value, $s_{i(k)}$ is the weight selection row vector for this location in the input space, β is the learning rate, C is the number of weights contributing to the CMAC response known as the generalization factor, and w is the new weight vector after update. In words, an error value is calculated between the desired network response d and the response produced by the existing network weights for that location according to the CMAC response law. This error is scaled by the learning rate and this new correction is added to each of the C contributing weights. This update law is made local by the presence of $s_{i(k)}$ premultiplying the error.

Only the weights contributing to the CMAC response at this location are corrected. This simple algorithm results in very rapid network performance as compared with global updating networks like the multi-layer perceptron.

4.2.2 Optimization of New Weights

The cost function for this problem is defined in Eqn 4.4 and the solution for batch mode weight updating is given by Eqn 4.6 and repeated here for convenience.

$$\underline{w} = \underline{Q}^{-1}\underline{S}^{t}(\underline{S}\underline{Q}^{-1}\underline{S}^{t})^{-1}\underline{Z}$$

For the case where the penalty matrix $\underline{Q}$ is the identity matrix, this solution takes the form of the pseudoinverse of the selection matrix $\underline{S}$ for the underdetermined case and is the minimum length solution and optimum in the least squares sense.

This rule is a batch mode weight assignment rule meaning that all the data are required in order to calculate the weight values and the weight values themselves are assigned as one operation. This is in contrast to the CMAC rule which is sequential and iterative in nature. Nevertheless, certain properties can be extrapolated from the study of the matrix $\underline{Q}^{-1}\underline{S}^{t}(\underline{S}\underline{Q}^{-1}\underline{S}^{t})^{-1}$ and will be presented later in this section.

4.2.3 Optimization of Weight Adjustments

The cost function for this problem is defined as follows:

$$J = \frac{1}{2}\underline{\Delta w}^{t}\underline{Q}\underline{\Delta w} + \underline{\lambda}^{t}(\underline{S}(\underline{w}_{o} + \underline{\Delta w})) - \underline{Z}) \qquad \text{Eqn 4.10}$$

The solution to the minimization of Eqn 4.10 is:

$$\underline{\Delta w} = \underline{Q}^{-1}\underline{S}^{t}(\underline{S}\underline{Q}^{-1}\underline{S}^{t})^{-1}(\underline{Z} - \underline{S}\underline{w}_{o}) \qquad \text{Eqn 4.11}$$

$$\underline{w} = \underline{w}_{o} + \underline{\Delta w} \qquad \text{Eqn 4.12}$$

This solution is much closer to the Albus CMAC update law especially when the

penalty matrix $\underline{Q}$ is the identity matrix. Again the solution takes the form of the pseudoinverse and in this case the weight update is minimum length. Unlike the Albus CMAC weight update law, the impact of adding a new data point is not limited to altering only the C selected weights nor is the same correction applied to all the weights. When the penalty matrix $\underline{Q}$ is set to minimize the difference between adjacent weights in order to enforce a smoothness penalty, the effect of new information can extend for many generalization distances from the selected weights. Again, this is a batch update rule and is therefore fundamentally different from the CMAC algorithm.

4.3 Results of Batch Mode Optimization Experiment in 1 Dimension

A weight matrix with 26 weights was created using MATLAB® and the weights were trained to approximate a parabola using the standard CMAC update law with a learning rate of 1. The training set consisted of every third data point excluding the endpoints. The CMAC generalization parameter was set to 5 meaning that 5 weights were added together to produce the network response. This setting for the generalization also guaranteed that the training for individual data points would overlap by 1 weight in the weight space. The weights were arranged in a 1-to-1 correspondence with the 26 data points of the parabola so that movement in the direction of increasing sample number translates to shifting receptive fields in the CMAC. No hashing was used. The arrangement is illustrated in figure 4.1.

4.3.1 Function Learning

Figure 4.2 is a plot of the parabola trained into the CMAC for this experiment. Five training passes were sufficient for this experiment. Figure 4.3 shows plots of the network response at all possible input points for two different network update laws. The

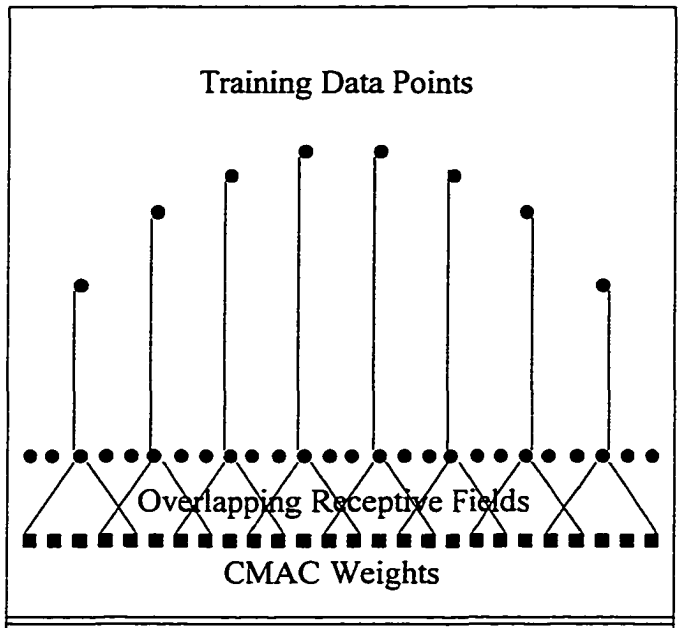


Figure 4.1: Relationship between training data and CMAC weights.

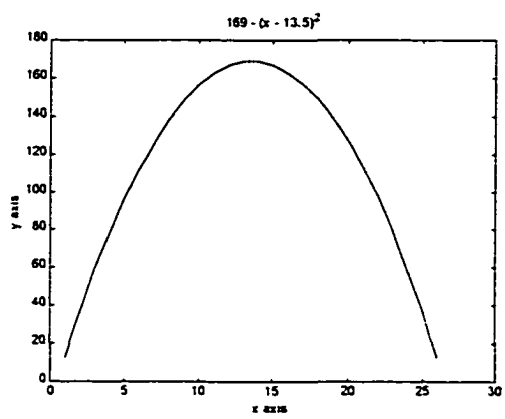


Figure 4.2: A plot of the parabola being trained into the CMAC. Every third data point was used for training

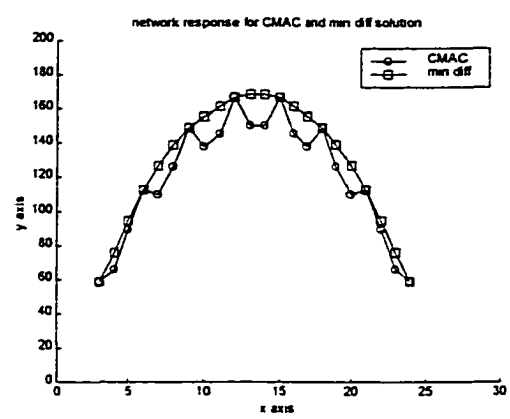


Figure 4.3: Plots of the network response at all possible points.

jagged response is produced by the traditional CMAC update law. The function approximation is clearly very good at the eight training points and gets progressively worse as the distance from the queried point to the trained data increases. The smooth response is produced by a minimum weight difference law where $\alpha = 1.0$ and $\beta = 0.1$ in Eqn 4.7. The network response is a much closer match to the desired parabola despite the sparse train-

network response is a much closer match to the desired parabola despite the sparse training data. This update law, in addition to being a batch mode law where all the data are used once in a single update, is also a globally updating law since all the weights are modified at once. Figure 4.4 shows plots of the network weights for the two update laws.

4.3.2 Weight Smoothing and Derivative Performance

Figure 4.5 shows plots of the two point derivative of the network response. The

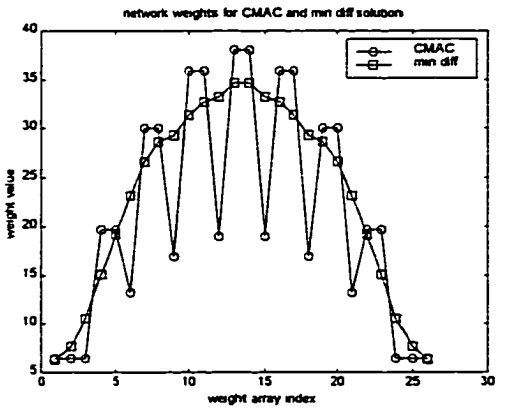


Figure 4.4: Plots of network weights

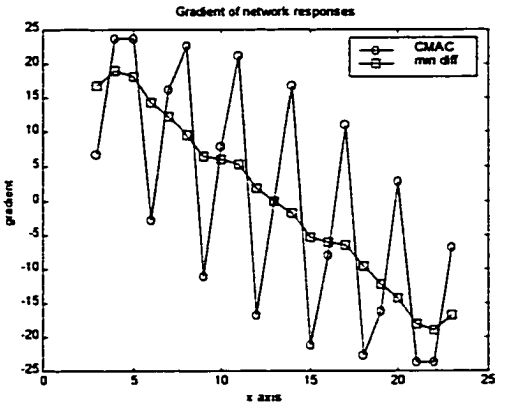


Figure 4.5: Plots of the derivative of the network response at all possible points.

jagged curve is the derivative of the response for the standard CMAC update law and the smooth curve is the derivative of the response for the minimum weight difference law. The derivative approximation is dramatically improved with the minimum weight difference update law.

4.4 Results of Batch Mode Optimization Experiment in 2 Dimensions

The one-dimensional example above, although illustrative, has several limitations which make it the trivial case. The concept of receptive field center placement has no meaning in the one-dimensional case. Receptive fields are placed in a linear fashion along

the data array with a receptive field center at every point in the quantized input space. There are always more weights (in unhashed and matrix representations) than there are states in the input space. The weight selection matrix is always full rank. None of these statements is true in the two-dimensional case. This makes the two-dimensional case the only case with all the problems and features of the CMAC which can be easily visualized using standard two- and three-dimensional visualization aids.

A parabola was again used as the desired function. The domain of the function was defined as $|X| \leq 10$, $|Y| \leq 10$ with the function itself defined as $f(X, Y) = X^2 + Y^2$. A plot of this function is shown below in figure 4.6.

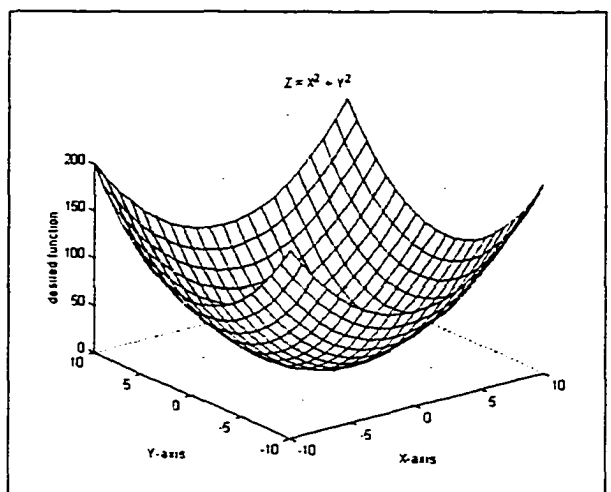


Figure 4.6: Desired function for two-dimensional experiment.

4.4.1 Function Learning

Figures 4.7 and 4.8 are examples of the matrix implementation Albus CMAC response at all points on the domain of the desired function depicted in figure 4.6. Figure 4.7 is for the case with weight smoothing and Figure 4.8 is for the case without weight smoothing. Figures 4.9 and 4.10 are examples of the An CMAC, with Gaussian receptive fields, response at all points on the domain of the desired function depicted in figure 4.6.

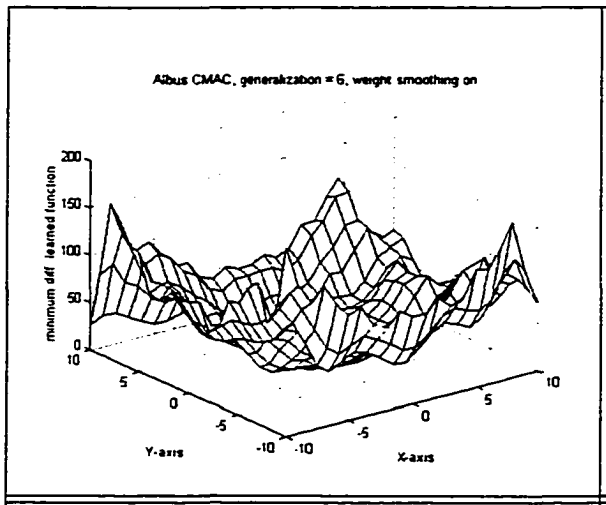


Figure 4.7: Matrix implementation, Albus law, network response, minimum length solution with additional minimum difference penalty.

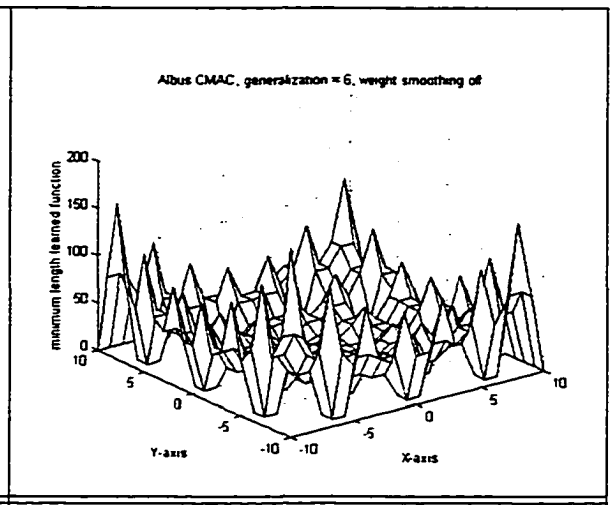


Figure 4.8: Matrix implementation, Albus law, network response, minimum length solution without additional minimum difference penalty.

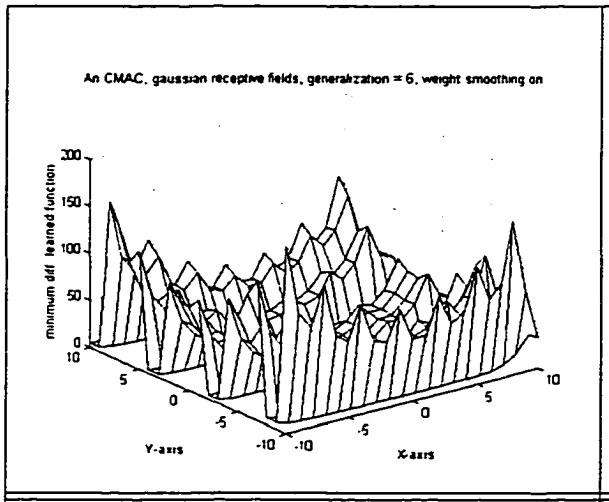


Figure 4.9: Matrix implementation, An law, network response, minimum length solution with additional minimum difference penalty.

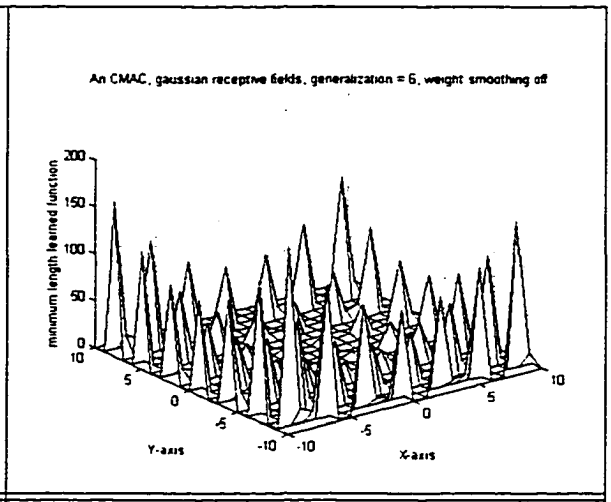


Figure 4.10: Matrix implementation, An law, network response, minimum length solution without additional minimum difference penalty.

Figure 4.9 is for the case with weight smoothing and figure 4.10 is for the case without weight smoothing. Both were trained with a generalization of six. Figure 4.9 indicates that there is a bias in the matrix implementation of this weight smoothing law. The training points were equally spaced in both the X- and Y-directions yet smoothing has occurred preferentially in one direction. This results because the mapping of the higher dimensional

weight space to the one-dimensional weight vector treats one dimension preferentially. Figures 4.11 and 4.12 are plots of the error between the desired function and the functions

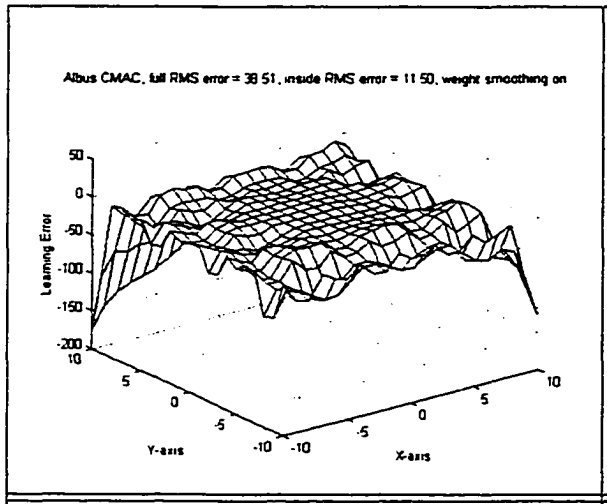


Figure 4.11: Function learning error, matrix implementation, Albus law, minimum length solution with additional minimum difference penalty.

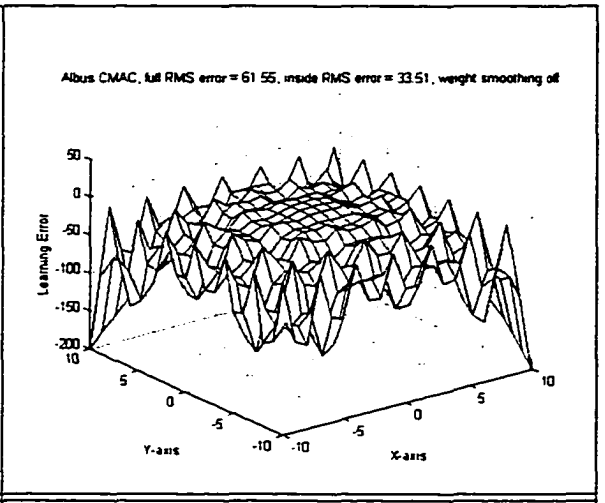


Figure 4.12: Function learning error, matrix implementation, Albus law, minimum length solution without additional minimum difference penalty.

of figures 4.7 and 4.8. The value of the RMS error is shown in the figure titles. The full RMS error is defined on the whole grid whereas the inside RMS error is defined on the grid excluding all points within three of the edges in an effort to eliminate contributions from edge effects.

Figures 4.13 and 4.14 are plots of the error surface defined as the difference between the desired function and CMAC responses on the whole grid for a network with the An receptive field placement with and without weight smoothing, respectively. Again, RMS errors for each surface are in the plot title.

Figure 4.15 is a plot of the full RMS error for an Albus type CMAC trained with and without weight smoothing. Figure 4.16 is a plot of the inside RMS error for an Albus type CMAC trained with and without weight smoothing. The smoothing case is optimiza-

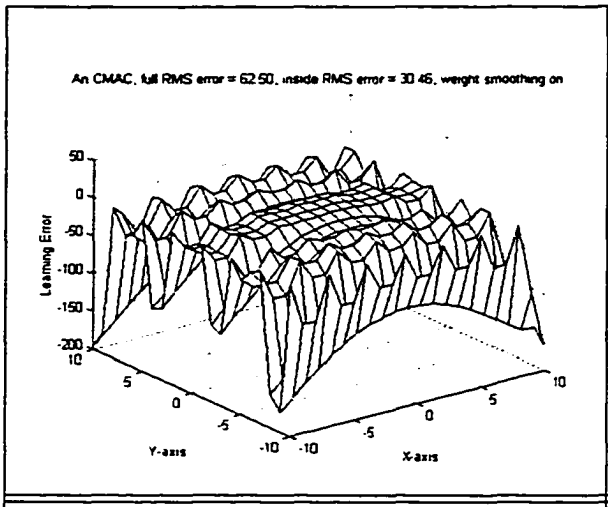


Figure 4.13: Function learning error, matrix implementation, An law, minimum length solution with additional minimum difference penalty.

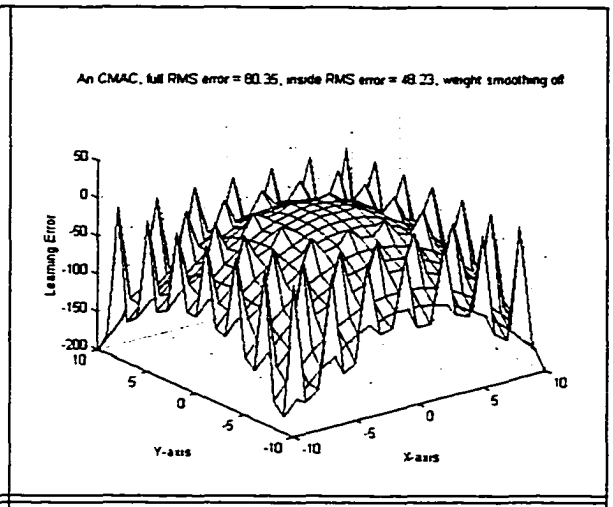


Figure 4.14: Function learning error, matrix implementation, An law, minimum length solution without additional minimum difference penalty.

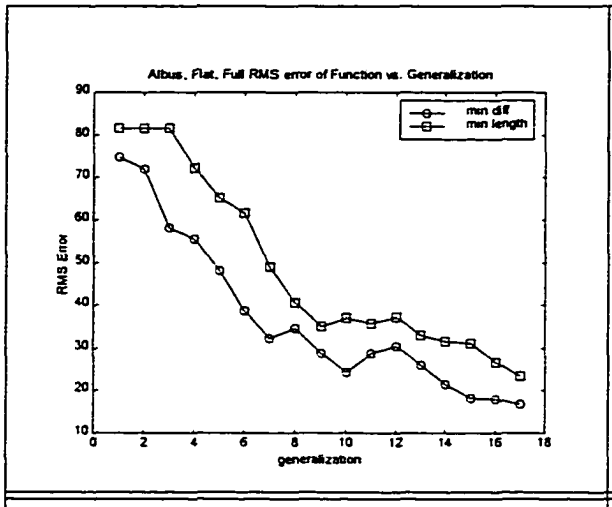


Figure 4.15: Full RMS learning error with and without weight smoothing, Albus law, training point spacing = 9.

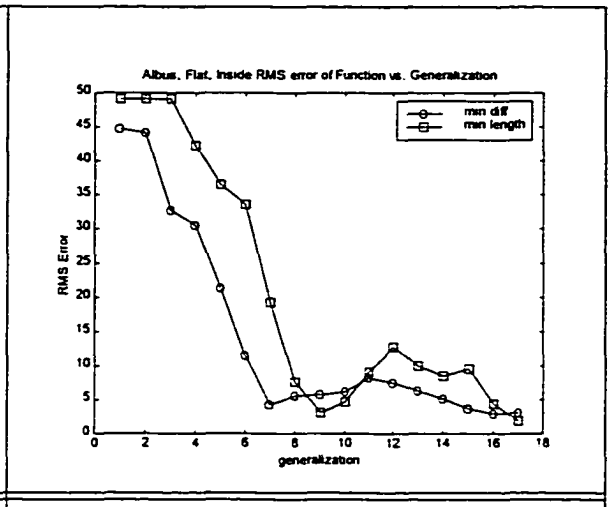


Figure 4.16: Inside RMS learning error with and without weight smoothing, Albus law, training point spacing = 9.

tion with the minimum weight difference penalty matrix. The case without smoothing is optimization for minimum weight length and has the identity matrix for a penalty matrix. The abscissa is the generalization parameter and ranges from one to seventeen. It is interesting to note that the first minimum for the Albus CMAC rms error occurs when the generalization is equal to the sample spacing. It is easier to see in some of the results from the

following chapter that successive minima occur when the generalization is equal to a multiple of the sample spacing as well.

Figure 4.17 is a plot of the full RMS error for an An type CMAC with Gaussian tapered receptive fields trained with and without weight smoothing. Figure 4.18 is a plot

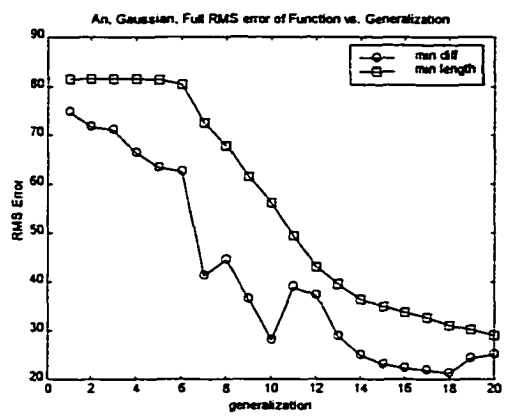


Figure 4.17: Full RMS learning error with and without weight smoothing, An law, training point spacing = 9.

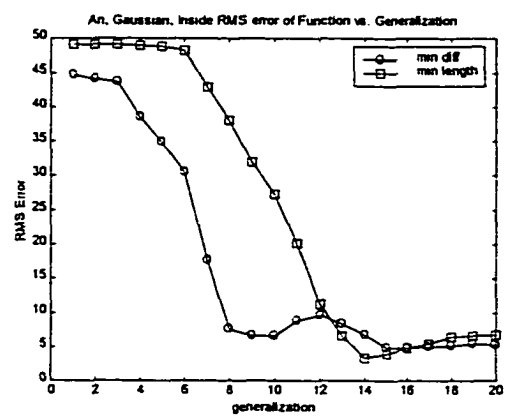


Figure 4.18: Inside RMS learning error with and without weight smoothing, An law, training point spacing = 9.

of the inside RMS error for an An type CMAC with Gaussian tapered receptive fields trained with and without weight smoothing. The abscissa is the generalization parameter and ranges from one to twenty.

These results show that, especially for values of the generalization parameter smaller than the training sample spacing, the optimum algorithm that incorporates a minimum weight difference penalty and, therefore, imposes a weight smoothing property on the learning rule has superior performance. This performance improvement occurs despite the bias built into the matrix implementation. As will be seen below, powerful insights can still be derived from these learning rules.

4.4.2 Weight Smoothing and Derivative Performance

Figure 4.19 is a close-up plot of a region of the CMAC weights for an Albus CMAC trained with and without weight smoothing. Figure 4.20 shows a close-up of a por-

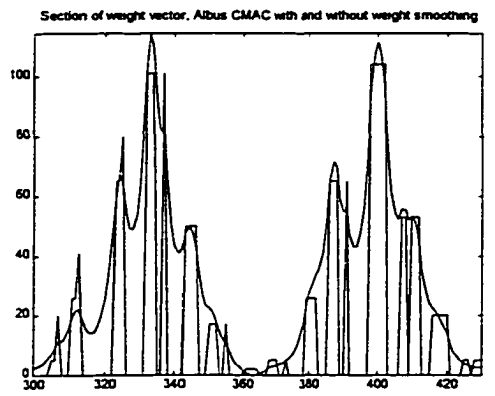


Figure 4.19: Network weights for minimum difference (smooth curve) and minimum length solutions

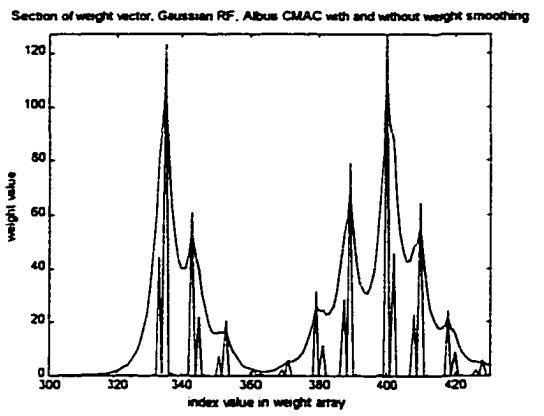


Figure 4.20: Network weights for minimum difference (smooth curve) and minimum length solutions.

tion of the CMAC weights for an An CMAC trained with and without weight smoothing. Both networks have the generalization parameter set to six. The weight smoothing is readily evident.

Two different tests were applied to evaluate derivative performance. First, the gradient of the CMAC response in the X- and Y-direction were compared to the corresponding gradients of the desired function. Figure 4.21 shows a plot of the RMS error in the X-gradient as a function of generalization for the Albus CMAC both with and without weight smoothing. Figure 4.22 shows a plot of the RMS error in the Y-gradient as a function of generalization for the Albus CMAC both with and without weight smoothing. Figure 4.23 and 4.24 show the corresponding X- and Y-gradient RMS error plots for an An CMAC.

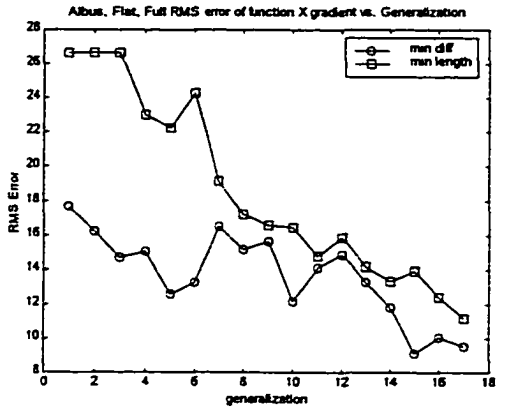


Figure 4.21: Full RMS error for X-gradient with and without smoothing, Albus law, training sample spacing = 9.

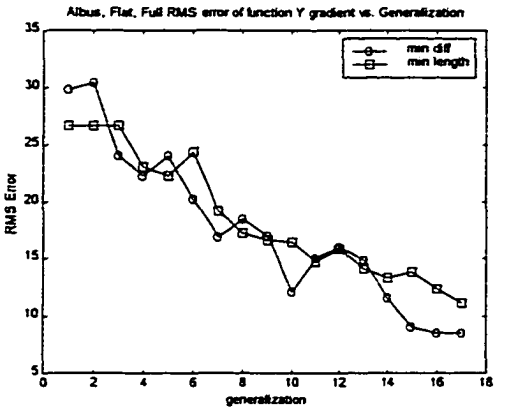


Figure 4.22: Full RMS error for Y-gradient with and without smoothing, Albus law, training sample spacing = 9.

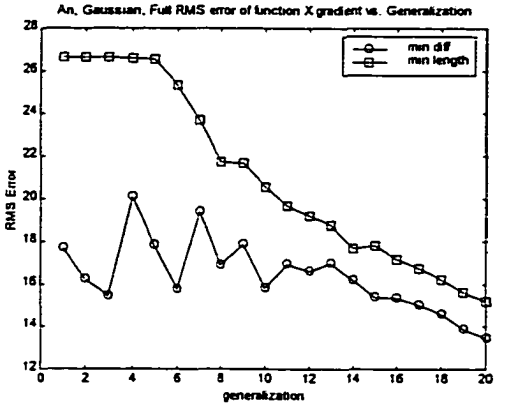


Figure 4.23: Full RMS error for X-gradient with and without smoothing, An law, training sample spacing = 9.

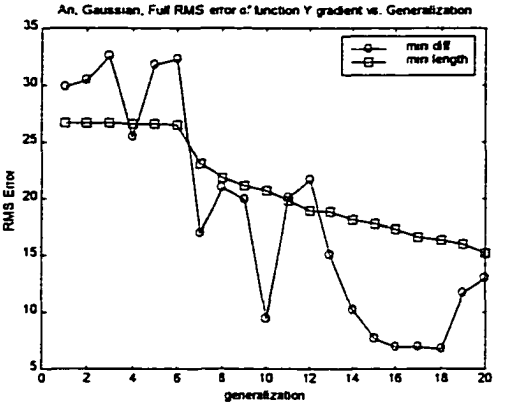


Figure 4.24: Full RMS error for Y-gradient with and without smoothing, An law, training sample spacing = 9.

Second, an approximation to the discrete Laplacian was used to evaluate the curvature properties of the CMAC responses relative to that of the desired function. The Laplacian of the desired function is a constant. Figure 4.25 is a plot of the RMS error between the Laplacian of the desired function and that of the Albus CMAC response for both the case when training is with weight smoothing and for the case when training is without smoothing. The abscissa is the generalization parameter which ranges from one to seven-

teen. Figure 4.26 is a plot of the RMS error between the Laplacian of the desired function

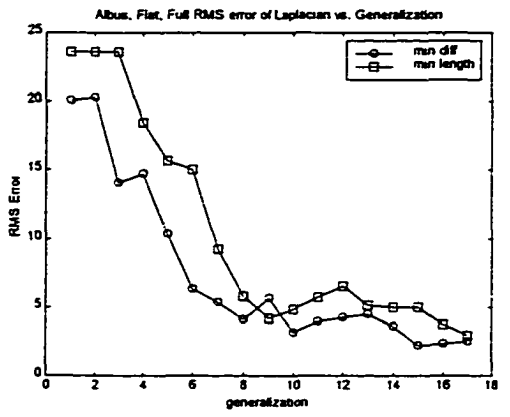


Figure 4.25: Full RMS error for discrete Laplacian with and without smoothing, Albus law, training sample spacing = 9.

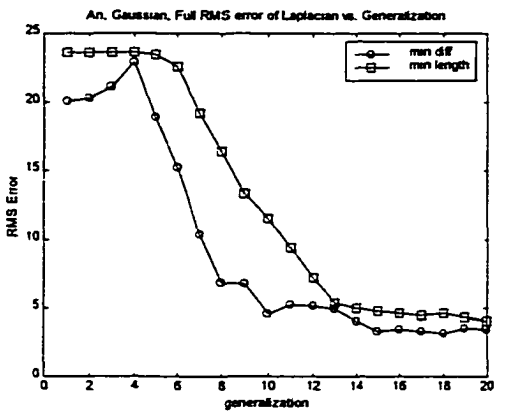


Figure 4.26: Full RMS error for discrete Laplacian with and without smoothing, An law, training sample spacing = 9.

and that of the An CMAC response for both the case when training is with weight smoothing and for the case when training is without smoothing. The abscissa is the generalization parameter which ranges from one to twenty. Figure 4.27 is a plot of $10*\log_{10}(\frac{\text{min\_length\_error}}{\text{min\_diff\_error}})$ for the Albus CMAC. Figure 4.28 is a plot of $10*\log_{10}(\frac{\text{min\_length\_error}}{\text{min\_diff\_error}})$ for the An CMAC with Gaussian receptive field weightings.

4.5 The Recursive Form for the Optimization Problem

Based on the results from the batch mode experiments, a search was begun for a recursive form for the optimal update law represented by Eqn 4.6 in order to more closely approximate the behavior of an actual CMAC. Taking the solution of Eqn 4.6 and adding a new data point yields:

$$\underline{w}_{k+1} = \underline{\underline{Q}}^{-1}\begin{bmatrix} \underline{S} \\ \underline{s}_{i(k)} \end{bmatrix}^t \left(\begin{bmatrix} \underline{S} \\ \underline{s}_{i(k)} \end{bmatrix} \underline{\underline{Q}}^{-1} \begin{bmatrix} \underline{S} \\ \underline{s}_{i(k)} \end{bmatrix}^t \right)^{-1} \begin{bmatrix} \underline{Z} \\ \underline{z}_k \end{bmatrix} \qquad \text{Eqn 4.13}$$

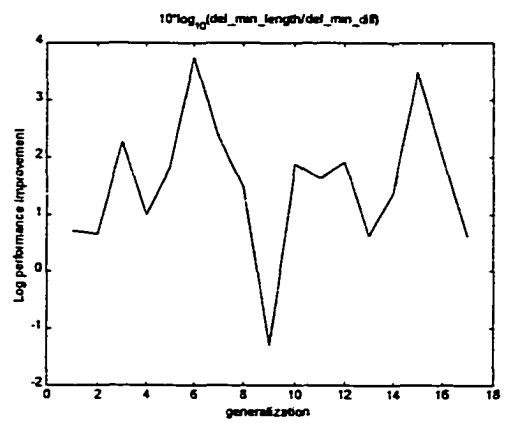


Figure 4.27: Log performance improvement factor. Values greater than zero indicate improved performance with weight smoothing. Vertical axis is in dB.

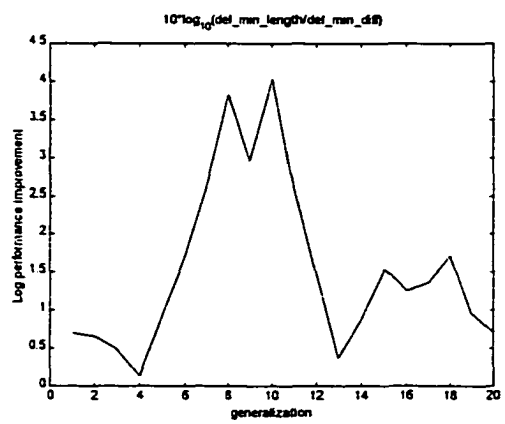


Figure 4.28: Log perfromance improvement factor. Values greater than zero indicate improved performance with weight smoothing. Vertical axis is in dB.

The lowercase letters with subscripts indicate the new sample being trained in at time k, and $\underline{w}_{k+1}$ is the new set of optimal weights. By taking advantage of the partitioned form of the matrices [18], the final form for the solution is:

$$\underline{w}_{k+1} = (\underline{Q}^{-1}\underline{S}^t\underline{B}_1 + \underline{Q}^{-1}\underline{s}^t_{i(k)}\underline{B}_3)\underline{S}\underline{w}_k + (\underline{Q}^{-1}\underline{S}^t\underline{B}_2 + \underline{Q}^{-1}\underline{s}^t_{i(k)}\underline{B}_4)z_k \qquad \text{Eqn 4.14}$$

For ease of reference Eqn 4.14 is rewritten as follows:

$$\underline{w}_{k+1} = \text{Wscale}*\underline{w}_k + \text{Zscale}*z_k \qquad \text{Eqn 4.15}$$

where:

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} = \begin{bmatrix} S\underline{Q}^{-1}S^t & S\underline{Q}^{-1}s_{i(k)}{}^t \\ s_{i(k)}\underline{Q}^{-1}S^t & s_{i(k)}\underline{Q}^{-1}s_{i(k)}{}^t \end{bmatrix} \qquad \text{Eqn 4.16}$$

and

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}^{-1} = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} = \begin{bmatrix} (A_1 - A_2A_4^{-1}A_3)^{-1} & -A_1^{-1}A_2(A_4 - A_3A_1^{-1}A_2)^{-1} \\ -A_4^{-1}A_3(A_1 - A_2A_4^{-1}A_3)^{-1} & (A_4 - A_3A_1^{-1}A_2)^{-1} \end{bmatrix}$$

The Wscale matrix forms a linear combination of the old weights to make up part

of the new weights. In the Albus type CMAC, this matrix is the identity matrix. The Zscale term represents the contribution of the new data point to each of the weights. The experiment has all weights initially zero and certain training points are visited in random order exactly once. The CMAC learning rate is set to 1. For the Albus CMAC the expected form of the Zscale vector is a group of C points with the same constant value and zeros for the rest of the elements. The expected form for the Wscale matrix is the identity matrix.

4.6 Sequential Update -- 1-Dimensional Experiment

The sequential update experiment was similar to the batch update experiment described above. The size of the weight vector has been expanded to 50 weights. The function to be learned was a parabola with data points arranged in a one-to-one correspondence with CMAC memory locations. The parameters that were varied during the experiment were the receptive field shape, the sample spacing for training, and the generalization width. CMAC learning rate was always unity. The sequential update law was always used once there were two or more samples available and the Q matrix parameters were $\alpha = 1.0$ and $\beta = 0.1$. The batch mode law was used to train in the first sample. Training data were presented in random order. Values of both the Wscale matrix and the Zscale vector were monitored.

Figure 4.29 shows a plot of the Zscale vector for the case where the first two data points being trained into the CMAC are far apart in the input space. The weight index axis indicates the indexed location of the accessed group of weights in the weight vector. A feature to notice here is that the algorithm spreads information out into the weight space much farther than the generalization value of 2 would suggest. One might view this as a

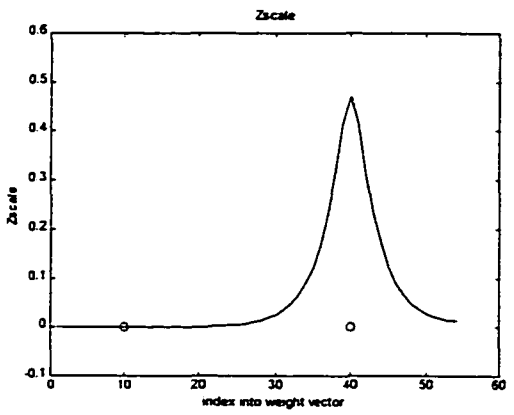


Figure 4.29: Plot of Zscale vector when the first two training points were widely separated in the input space. Receptive field: flat, generalization: 2, sample spacing: 3, point being trained: 39.

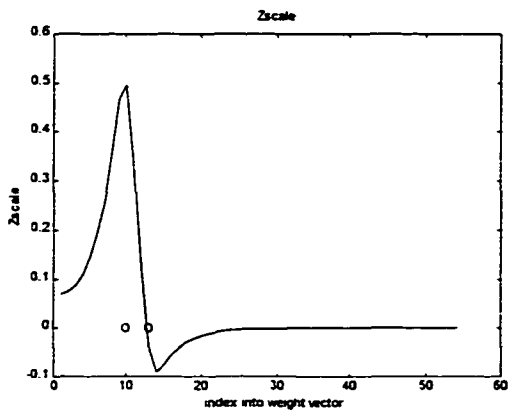


Figure 4.30: Plot of Zscale vector when first two training points were close in the input space. Receptive field: flat, generalization: 2, sample spacing: 3, point being trained: 9.

way of improving smoothness and generalization by extrapolation of the training data. Figure 4.30 is a plot of the Zscale vector for the case where the first two data points are near each other in the input space. Note how the shape of the new update curve is such that the new data point makes zero contribution to the location corresponding to the previously trained data point. In essence, the algorithm takes into account how much information is in the vicinity of the current training point and alters the shape of the new sample weighting function accordingly. These two features, extended generalization and compensating for previously trained data in the vicinity of a new update, will be discussed in greater detail in the section of the two-dimensional problem.

Figure 4.31 is a plot of the Zscale vector for the final training point in the series. For this flat receptive field shape, a new data point can affect almost all the weights in the network. The row of Wscale corresponding to sample number 12 is identically zero indicating that the update should come from the new data point. Figure 4.32 shows a plot of

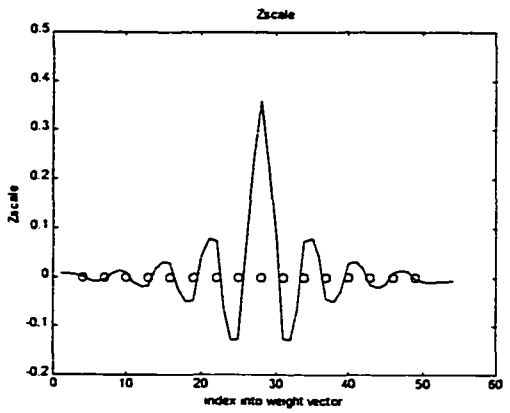


Figure 4.31: Plot of Zscale vector for the last training point in the series. Receptive field: flat, generalization: 2, sample spacing: 3, point being trained: 27.

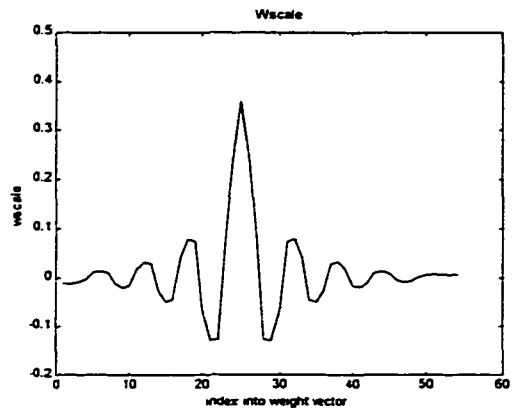


Figure 4.32: Plot of the Wscale row vector corresponding to sample point #30 during the training of sample point # 27.
Receptive field: flat, generalization: 2, sample spacing: 3.

the row of the Wscale matrix corresponding to new weight number 15 (an adjacent training point) showing how the value stored in this new weight will depend upon the previous values of all the other weights in the network. In essence such an optimum CMAC would have to update globally on each training pass in order to satisfy the smoothness and minimum weight size constraints. Figure 4.33 shows another plot of the Zscale vector for the final data point in the training series. In this case, however, the CMAC receptive field was set to a Gaussian shape. This was accomplished by changing the rows of the $\underline{S}$ matrix and the $s_{i(k)}$ vector so that the entries have a Gaussian profile instead of the unity values for the flat receptive field. In this case the contribution of a new weight reaches out only a finite distance into the weight space. The plot in figure 4.34 shows that in this case the new weights are a linear combination of only nearby weights. Local updating behavior might be much easier to approximate with a CMAC using Gaussian receptive fields. Figures 4.35 and 4.36 show the effects of increasing the size of the generalization from 2 to 5

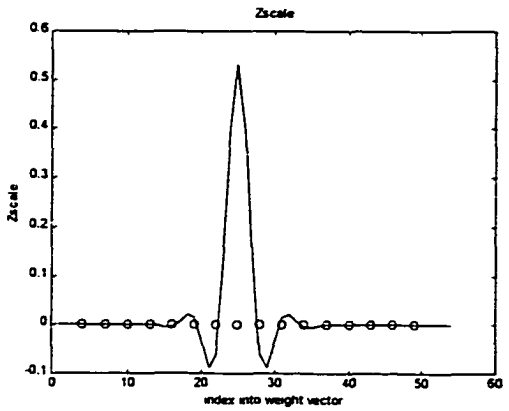


Figure 4.33: Plot of Zscale vector for the last training point in the series. Receptive field: Gaussian, generalization: 2, sample spacing: 3, point being trained: 24.

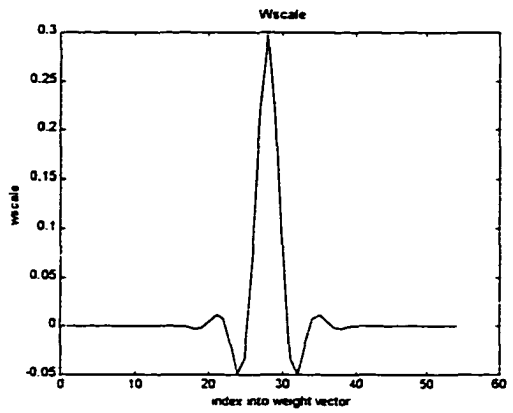


Figure 4.34: Plot of the Wscale row vector corresponding to sample point #27 during the training of sample point # 24.
Receptive field: Gaussian, generalization: 2, sample spacing: 3.

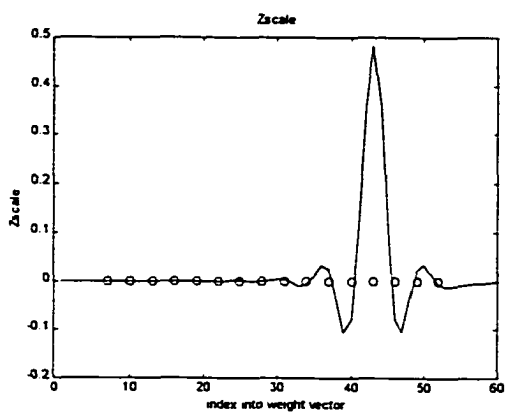


Figure 4.35: Plot of Zscale vector for the last training point in the series. Receptive field: Gaussian, generalization: 5, sample spacing: 3, point being trained: 42.

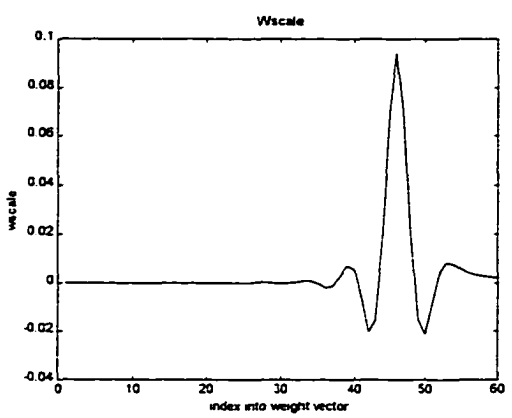


Figure 4.36: Plot of the Wscale row vector corresponding to sample point #45 during the training of sample point # 42.
Receptive field: Gaussian, generalization: 5, sample spacing: 3.

(the central weight plus 2 weights on either side of it). All the other parameters were left unchanged from the first experiment with Gaussian receptive fields. There appears to be almost no difference between these scaling values and those presented in figures 4.33 and

4.34. Had there been no pretrained data in the vicinity of this final training point, the curve would have been much more broad. The update law's compensation nearby trained information causes the curve to be so similar to the previous example. This compensation effect will be explained in more detail in the following section.

Figures 4.37 and 4.38 show the effect of doubling the sample spacing parameter

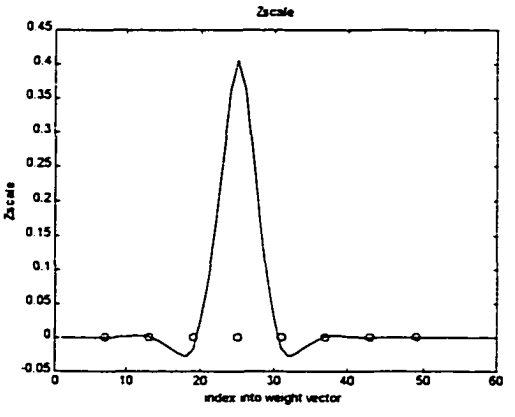


Figure 4.37: Plot of Zscale vector for the last training point in the series. Receptive field: Gaussian, generalization: 5, sample spacing: 6, point being trained: 24.

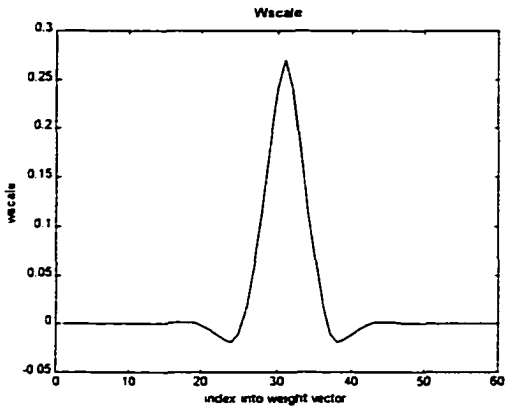


Figure 4.38: Plot of the Wscale row vector corresponding to sample point #30 during the training of sample point # 24.
Receptive field: Gaussian, generalization: 5, sample spacing: 6.

and leaving the others unchanged. The weighting functions are simply expanded so that their zeros again pass through the locations of the training data. This again demonstrates the self-limiting behavior of this optimum weight adjustment scheme.

4.7 Sequential Update -- 2-Dimensional Experiment

Since the 2-dimensional case is representative of general CMAC characteristics it will be used as the case from which to derive the relevant features for the new weight smoothing CMAC weight adjustment law. The process begins with an examination of the

components comprising the sequential update law repeated here for convenience.

$$\underline{w}_{k+1} = (\underline{\underline{Q}}^{-1}\underline{\underline{S}}^t\underline{\underline{B}}_1 + \underline{\underline{Q}}^{-1}\underline{s}^t_{i(k)}\underline{B}_3)\underline{\underline{S}}\underline{w}_k + (\underline{\underline{Q}}^{-1}\underline{\underline{S}}^t\underline{\underline{B}}_2 + \underline{\underline{Q}}^{-1}\underline{s}^t_{i(k)}\underline{B}_4)y_k$$

Rewriting to separate terms produces:

$$\underline{w}_{k+1} = \underline{\underline{Q}}^{-1}\underline{\underline{S}}^t\underline{\underline{B}}_1\underline{\underline{S}}\underline{w}_k + \underline{\underline{Q}}^{-1}\underline{s}^t_{i(k)}\underline{B}_3\underline{\underline{S}}\underline{w}_k + \underline{\underline{Q}}^{-1}\underline{\underline{S}}^t\underline{\underline{B}}_2 y_k + \underline{\underline{Q}}^{-1}\underline{s}^t_{i(k)}\underline{B}_4 y_k$$

Each of these four terms will now be examined individually.

4.7.1 The smoothing matrix Q inverse

Before looking at the individual terms in the weight adjustment law it is instructive to examine the inverse of the penalty matrix $\underline{\underline{Q}}$ since it is a premultiplier for all the terms in the weight adjustment law. Like $\underline{\underline{Q}}$, $\underline{\underline{Q}}^{-1}$ is symmetric and positive definite. Figure 4.39

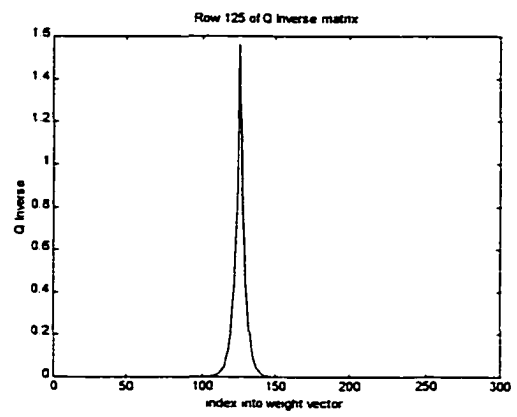


Figure 4.39: Plot of one row of the $\underline{\underline{Q}}^{-1}$ matrix.

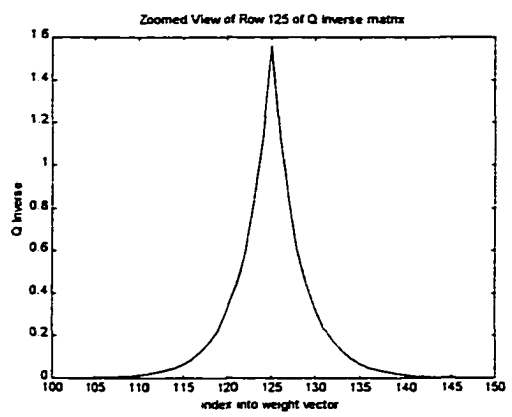


Figure 4.40: Zoomed view of the smoothing function from the $\underline{\underline{Q}}^{-1}$ matrix.

shows a plot of a typical row of the $\underline{\underline{Q}}^{-1}$ matrix while figure 4.40 zooms in on the smoothing function. The functional form of this curve is

$$\frac{1.5617}{1.36426^{|x-x_0|}} \qquad \text{Eqn 4.17}$$

This functional shape will form the basis for the weighting function in the new

CMAC weight adaptation rule.

4.7.2 Term 1

Term 1 is defined as

$$\underline{\underline{Q}}^{-1}\underline{S}^{t}\underline{B}_{1}\underline{S} \qquad \text{Eqn 4.18}$$

and premultiplies $\underline{w}_k$. Components of term1 are: $\underline{S}$ which is the matrix of all previously trained selection matrices; $\underline{\underline{Q}}^{-1}$; and $\underline{B}_1 = (A_1 - A_2 A_4^{-1} A_3)^{-1}$. The A_i components are defined in equation 4.14. Term 1 defines the averaging and smoothing function on the previously trained weights. The new weights, before adjustment, are defined as a scaled sum of old weights. Figure 4.41 is an intensity plot of a portion of the term 1 matrix during the early stages of learning. The contribution of $\underline{\underline{Q}}^{-1}$ can clearly be seen in the intensity of the active regions. This plot is oriented in the natural matrix fashion with coordinate (1, 1) in the upper lefthand corner of the plot.

4.7.3 Term 2

Term 2 is defined as $\underline{\underline{Q}}^{-1}\underline{s}^{t}_{i(k)}\underline{B}_3\underline{S}$ and premultiplies $\underline{w}_k$. Components of term 2 are: $\underline{S}$ which is the matrix of all previously trained selection matrices; $\underline{s}^{t}_{i(k)}$ which is the selection vector for the new data point; $\underline{\underline{Q}}^{-1}$; and $\underline{B}_3 = -A_4^{-1} A_3 (A_1 - A_2 A_4^{-1} A_3)^{-1}$. Term 2 identifies locations in the weight space where the newly selected weights interact in some way with previously trained information. Due to the extent in the weight space of the smoothing function introduced by $\underline{\underline{Q}}^{-1}$ interactions can occur with weights at seemingly great distances in the weight space, however the magnitude of contributions from term 2 are often very small due to the decaying nature of the smoothing function. Figure 4.42 shows an intensity plot of term 2 during the early stages of learning. The coordinates of the new weights being trained are 60, 61, 62, 63. Their location can be seen in figure 4.41.

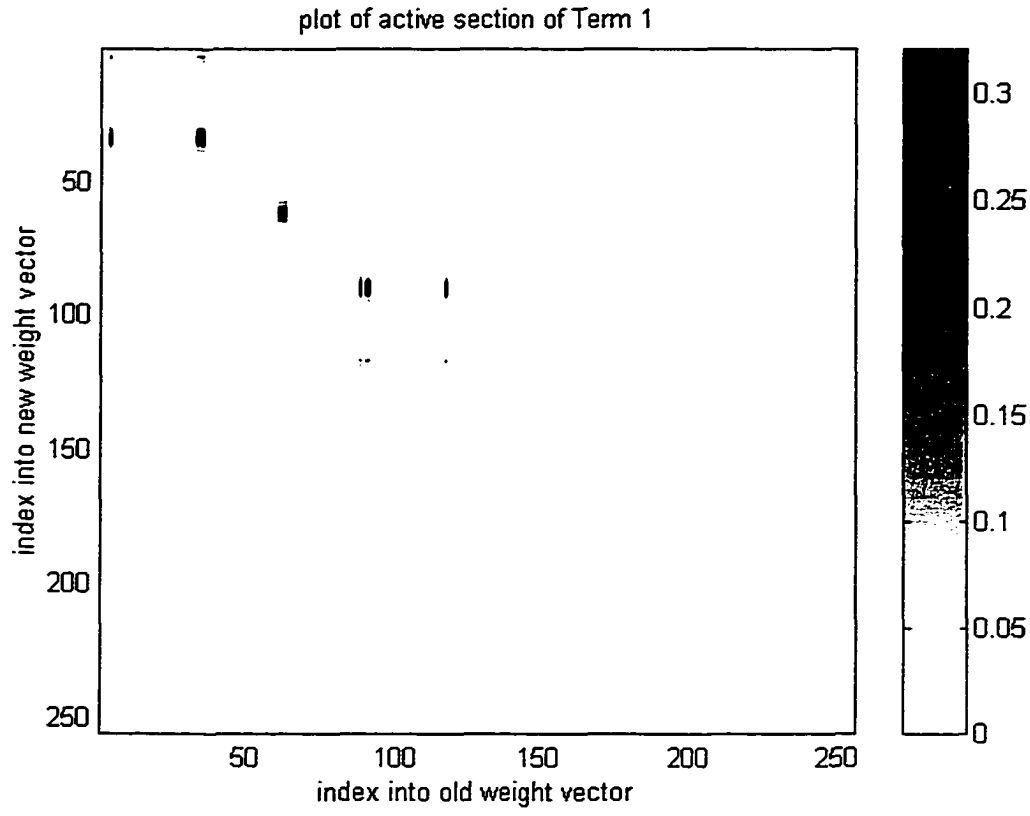


Figure 4.41: Wscale matrix before training fourth data point. Network generalization = 4, 3 samples have been trained previously, indices of previously trained weights are: 4, 33, 34, 35, 60, 61, 62, 63, 88, 90, 91, 117

The tails of the smoothing functions are interacting in this case as can be seen in figure 4.41. This is the reason the amplitude of the correction is so small -- fully four orders of magnitude below other signal levels. When previously trained weights are close to the weights accessed by the new data point, this interaction can be quite strong. When interaction is strong, the effect is to precompensate information already stored in the weights where the interaction will occur to minimize the effect of the interaction on the stored information. When training is dense, this compensation effect can ripple out a great distance into the weight space.

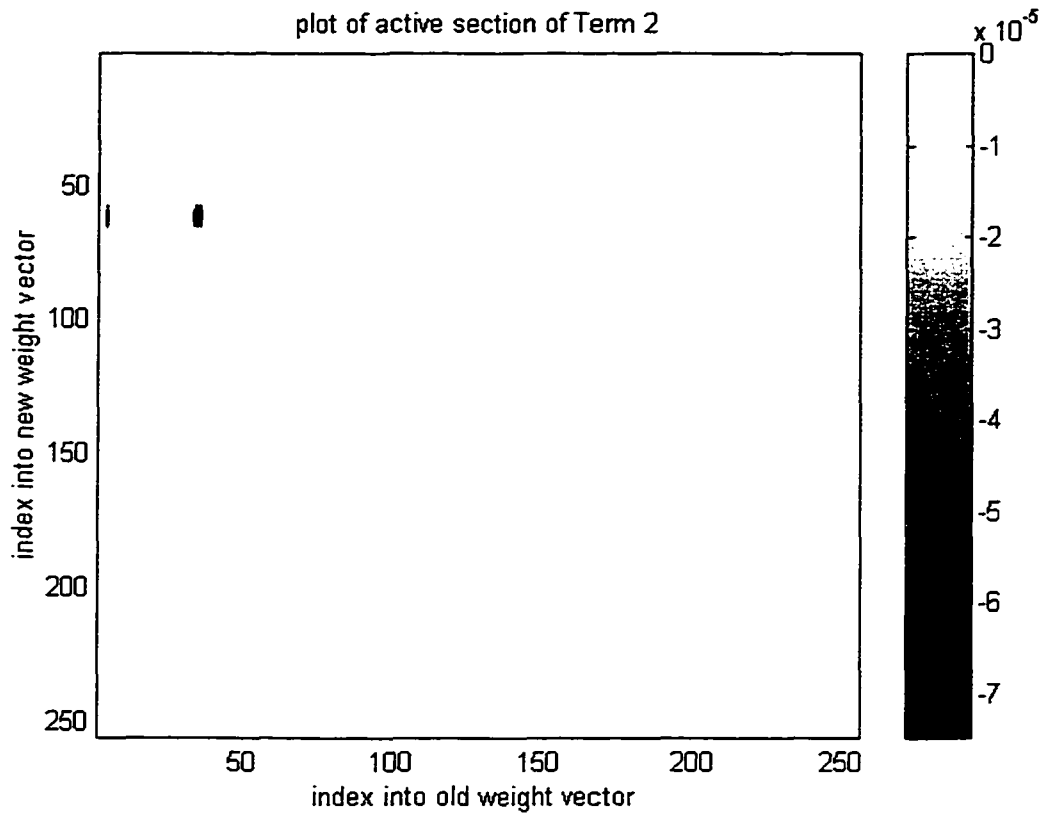


Figure 4.42: Correction to linear combination of old weights prior to application of new data point is a function of interaction between first and second trained points. Sample spacing is large enough to prevent strong interaction.

4.7.4 Term 3

Term 3 is defined as $\underline{\ }^{-1}\underline{S}^{t}\underline{B}_2$ and premultiplies z_k. Components of term 3 are: $\underline{S}$ which is the matrix of all previously trained selection matrices; $\underline{Q}^{-1}$; and $\underline{B}_2 = -A_1^{-1}A_2(A_4 - A_3A_1^{-1}A_2)^{-1}$. Term 3 is the transpose of term 2 and identifies corrections to the new weight adjustment arising from interactions between the new weights and previously trained information. Figure 4.43 is a plot of term 3 under the same conditions as those under which figure 4.42 was generated. It clearly shows a weak interaction with the data trained on the previous pass and stored in locations 4, 33, 34, and 35. In this case, when interaction is strong, the learning function defined by term 4 is truncated in the

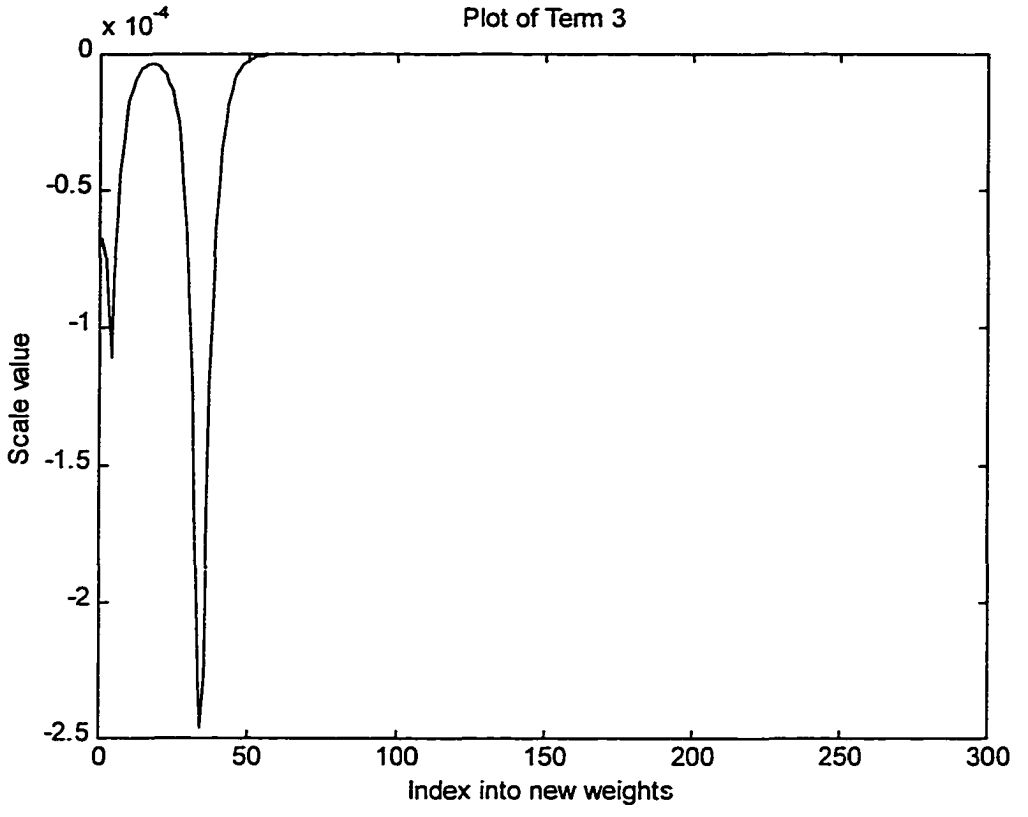


Figure 4.43: Correction to new sample scaling is a function of interaction between first and second trained points. Sample spacing is large enough to prevent strong interaction.

direction of previously trained information in order to minimize the interference between previously trained data and the new training point.

4.7.5 Term 4

Term 4 is defined as $\underline{\underline{Q}}^{-1}\underline{s}^{t}_{i(k)}\underline{B}_4$ and premultiplies z_k. Components of term 4 are: $\underline{s}^{t}_{i(k)}$ which is the selection vector for the new data point; $\underline{\underline{Q}}^{-1}$; and $\underline{B}_4 = (A_4 - A_3 A_1^{-1} A_2)^{-1}$. The scalar term $\underline{B}_4$ is essentially $\frac{1}{A_4}$ unless there is significant interaction between the new sample's weights and previously trained information. If there were no smoothing and Albus receptive fields, $\underline{B}_4 \approx \frac{1}{C}$. Figure 4.44 is a plot of term 4 created under the same conditions as those for figure 4.43. The weights being trained are at

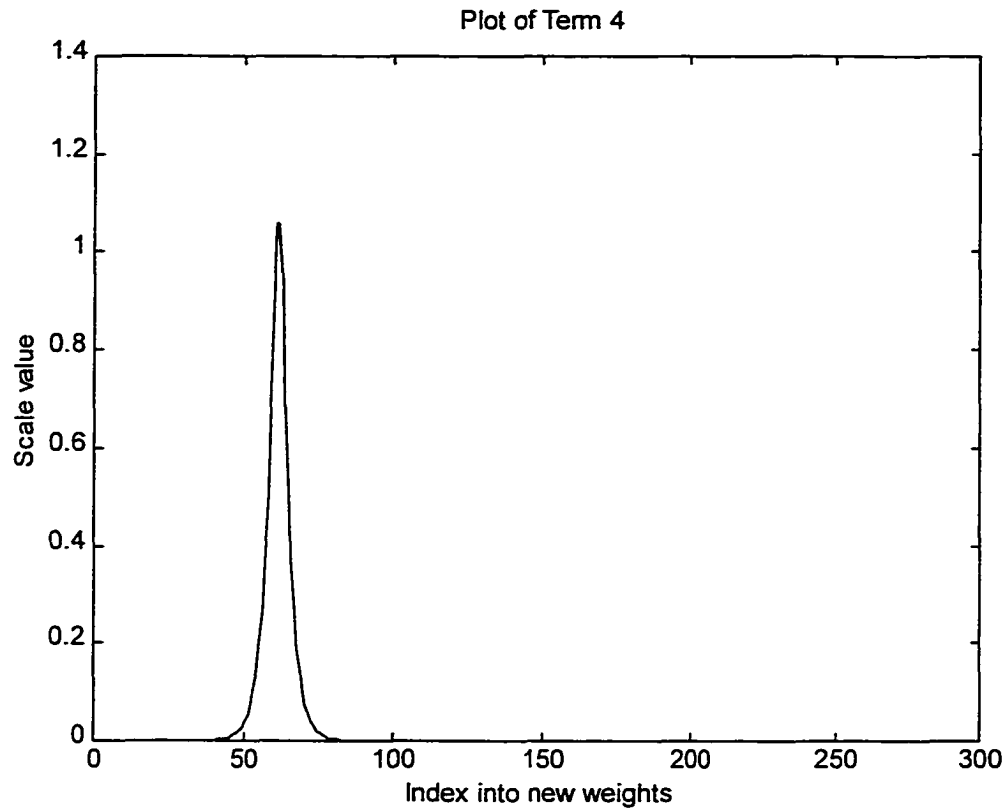


Figure 4.44: Scaling and distributed application of new data point is controlled primarily by $\underline{Q}^{-1}$.

locations 60, 61, 62, 63.

4.8 The New CMAC Weight Smoothing Law

Now that the components of the optimal matrix sequential weight smoothing law have been broken out and studied, it is possible to specify several new weight adjustment rules which progressively approach an approximation to the optimum matrix solution presented above.

4.8.1 New Rule 1

The simplest new learning rule is to implement the portion of the adjustment law represented by term 4. This update rule will successfully spread information out into the

weight space in a way that will greatly enhance learning and function approximation especially when training data are widely separated. Successful implementation of this rule requires a new kind of generalization in the CMAC. This is developed in the next chapter and leads to experiments using new rule 1. Using the technology available in the modified CMAC code developed under this research, rule 1 is implemented by moving $\underline{Q}^{-1}\underline{s}^{t}_{i(k)}\underline{B}_{4}$ back into the weight space and treating this function as a tapered receptive field. This new receptive field is defined to have a large area and dense support. The Super Generalization mechanism is described in detail in the next chapter. Performing rule 1 in the weight space rather than in the linear weight array removes any bias derived from the mapping from the n-dimensional weight space to the 1-dimensional weight array. This new weight update law does not explicitly perform weight smoothing. It spreads new training information out in a high density learning mode which will often provide much better coverage than other generalization techniques. The iterative training process in the CMAC actually performs a smoothing operation through the coupling of individual training points through common weights. This is greatly enhanced using rule 1.

4.8.2 New Rule 2

New rule 2 implements the functionality of term 1 as described above. This averaging function should also be promoted back to the weight space in order to be performed correctly. The smoothing operation is really a weight averaging procedure performed on groups of weights selected using the base generalization. For every weight in the Super Generalization cell, that weight and its nearest neighbor weights, as defined by the base generalization selection function, are averaged together. The average is scaled by the weighting function for the Super Generalization cell at that point and this scaled average is

set aside to become the new weight value at that location. This process is repeated for all the weights in the Super Generalization cell and then all the old weights are updated with the new values. This rule explicitly implements a weight smoothing operation.

4.8.3 New Rule 3

Rule 3 adds a measure of context sensitivity to the overall weight update process. The process involves sampling the weight space corresponding to state space points along each of the dimensions of the problem looking for previously trained data. The size of the Super Generalization cell and possibly its shape as well should then be varied to provide the best coverage for the new training data point while at the same time minimizing the loss of information previously trained into the network. Some amount of training overlap would be prescribed to preserve smoothness and continuity across the training boundary. For the original Albus CMAC the optimum amount of overlap occurred when the generalization size was equal to the sample spacing. This was pointed out in the discussion of figures 4.15 and 4.16. Rule 3 would only be used in conjunction with either rule 1 or rule 2 to improve performance. A network trained with all three rules would most closely approximate the optimal solution as defined by the constrained optimization problem described in this dissertation. Others certainly are possible.

Chapter 5

The CMAC Weight Smoothing Law

In order to implement the new weight adjustment law suggested by the optimum matrix solution, the CMAC algorithm must be modified so that it is possible to train with one value of the generalization parameter and remember with another value of the generalization parameter. The existing CMAC properties make it ineffective to train the network with one generalization value and then calculate network responses with a different value of generalization. The details of this issue are presented below along with the solution developed under this research. This new generalization technique, Super Generalization, has been implemented in C as part of the UNH CMAC code and is used to implement the new weight adjustment law developed in the previous section. Experimental results are presented.

5.1 Issues of Generalization and Super Generalization

It is worthwhile to examine how the current CMAC code actually allocates weights for generalization. Generalization is really about determining how many receptive fields are excited by a point in the input space and about how the receptive field centers are distributed in the input space. Figures 5.1 and 5.2 show the receptive field center allocation patterns for the two-dimensional Albus CMAC with two different values of the generalization parameter. Training points are marked by asterisks and the lower, left-hand

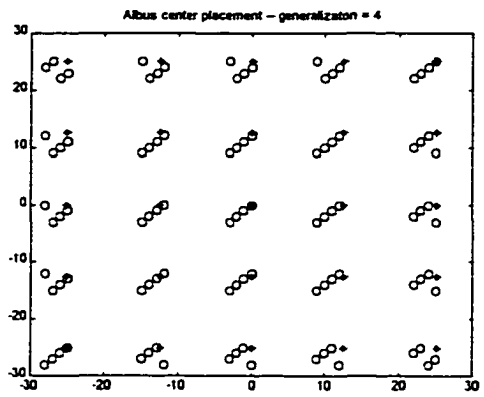


Figure 5.1: Receptive field center allocation pattern for Albus CMAC with generalization of 4. o's represent the lower left-hand corner of a receptive field and *'s represent training points.

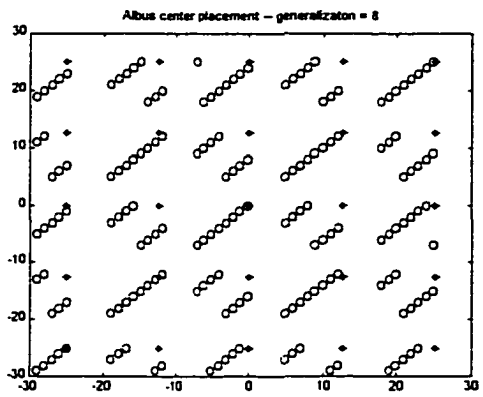


Figure 5.2: Receptive field center allocation pattern for Albus CMAC with generalization of 8. o's represent the lower left-hand corner of a receptive field and *'s represent training points.

corners of the receptive fields are marked by 'o's. It is characteristic of the Albus receptive field center placement strategy for the centers to be arranged along diagonals. For this two-dimensional example the receptive fields can be thought of as having square bases. For the single output CMACs considered in this research each receptive field has a single weight associated with it. The coordinates used to generate these plots are the raw coordinates of the receptive field corners and correspond to coordinates in the input space. No hashing or mapping to a linear memory array has been performed. Note that the patterns are different for the two different values of the generalization parameter. Specifically, the spacing between the diagonal groupings of receptive field centers is equal to the generalization size.

This section addresses the question of what happens if the CMAC is trained with one value of generalization while network responses are determined with a different value of generalization. The specific issue can be phrased as follows: since training with large

generalization spreads information out to a larger number of weights and makes the effects of the training available to more points in the nearby input space, why not train with large generalization to get good spread of information, particularly in the early training, and then use a smaller value of generalization, for better spatial resolution, to derive network responses and ultimately to refine the training as well.

Since the goal of this research is to improve the CMAC performance when training is sparse, it is instructive to explore how trained weights influence the network response for locations at which no training has occurred. Figures 5.3 and 5.4 show intensity plots of

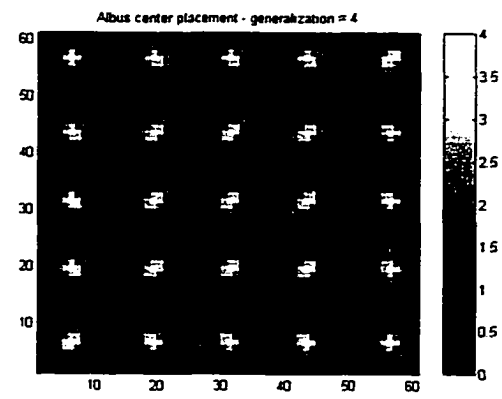


Figure 5.3: Learning support in Albus CMAC with generalization of 4.

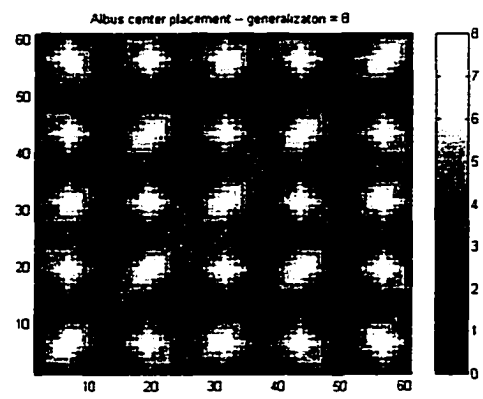


Figure 5.4: Learning support in Albus CMAC with generalization of 8.

how many trained weights contribute to the CMAC response at every point in the input space. The actual training points can be identified as the high intensity points in the plot. A regular pattern of support is visible. The square shape of the receptive field bases is clearly visible in these figures. Figures 5.5 and 5.6 show the pattern of receptive field center placement when the An heuristic placement strategy is used. The pattern of receptive field center placements is more uniform. This arrangement of receptive field centers helps to

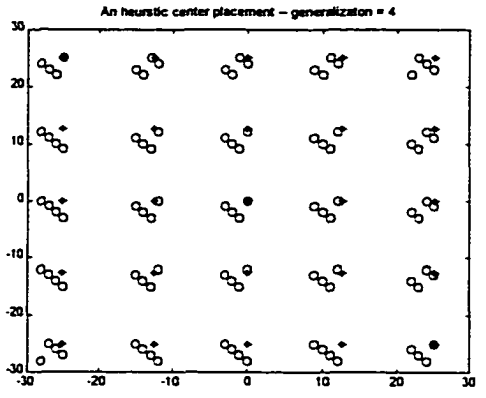


Figure 5.5: Receptive field center allocation pattern for An CMAC with generalization of 4. o's represent the lower left-hand corner of a receptive field and *'s represent training points.

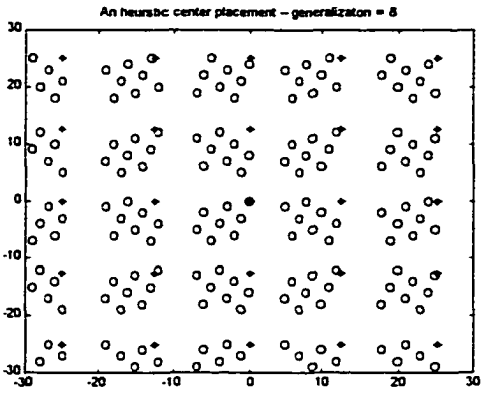


Figure 5.6: Receptive field center allocation pattern for An CMAC with generalization of 8. o's represent the lower left-hand corner of a receptive field and *'s represent training points.

reduce the problem of preferential generalization which can occur with the Albus placement strategy. Figures 5.7 and 5.8 show the support this strategy provides for nearby

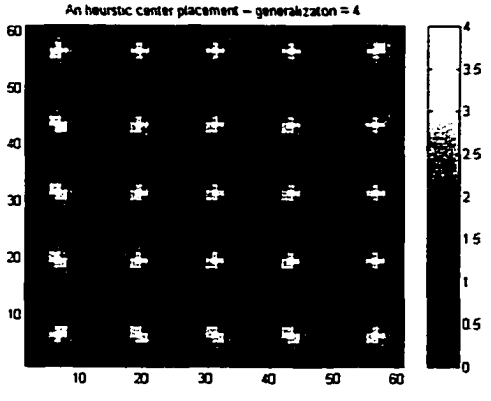


Figure 5.7: Learning support in An CMAC with generalization of 4.

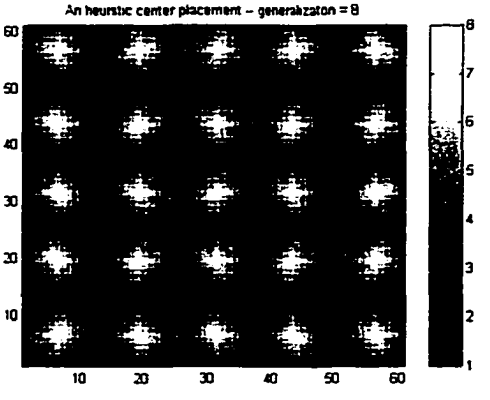


Figure 5.8: Learning support in An CMAC with generalization of 8.

untrained points in the input space. This receptive field center placement strategy provides more uniform support across the input space as can be seen from the intensity bar in figure

5.8. Since the minimum value in the plot shown in figure 5.8 is 1, there is no point in the input space shown where no weights contribute to the network response. Contrast this to figure 5.4 where there are large areas of the input space that are not covered by any trained weights.

Of course, the real issue is how different generalization patterns interact as this will determine the effectiveness of learning and remembering with different generalizations. Figure 5.9 shows the support provided using the Albus receptive field center placement strategy when training is done with a generalization of 8 and network responses are obtained using a generalization of 4. A large amount of the information trained into the network with the larger generalization is no longer available for generating network responses at the smaller generalization. Figure 5.10 shows the support provided using the

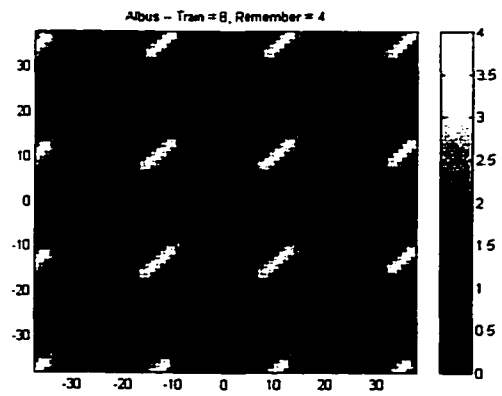


Figure 5.9: Plot of support provided using the Albus receptive field center placement strategy, training gen = 8, remember gen = 4.

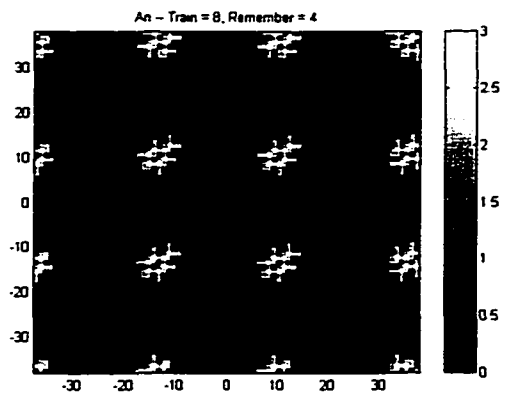


Figure 5.10: Plot of support provided using the An receptive field center placement strategy, training gen = 8, remember gen = 4.

An receptive field center placement strategy when training is done with a generalization of 8 and network responses are obtained using a generalization of 4. To understand why two

different generalizations interact so poorly, it is useful to imagine a two-dimensional, n-dimensional in general, weight space $\aleph$ where there are as many weights as there are points in the input space. Such a weight space grows exponentially with the input dimension. CMAC receptive field center allocation strategies seek to identify subsets of $\aleph$ which project uniformly onto each input dimension. Each value of generalization directs the selection of a different subset and it is often the case that these subsets have only weak intersections.

With the existing CMAC, generalization is the only method available to determine how widely the value of new training data is spread out in the weight space. A larger value of generalization means a larger number of weights are updated. It also means that the physical extent of the generalization region is larger. This is accomplished by increasing the spacing between the diagonals of receptive field centers. A larger generalization region results in more filtering of high frequency spatial variation in the input space than does a smaller generalization region. The optimal solution for the weight smoothing algorithm as developed in the matrix case requires that training be performed with a large generalization value, on the order of 35-40 for unity scaled minimum difference penalty, as defined by the inverse of the penalty matrix. Network responses can be determined using any generalization but typically much smaller than the training generalization, so in most cases each network will require successful operation with two different generalizations.

This research proposes an addition to the existing CMAC algorithm capable of supporting the previously incompatible requirements of training with a (sometimes) large generalization yet being able to generate network responses using a smaller generalization. Called Super Generalization, this new generalization technique trains a large region

of the weight space using the receptive field center allocation pattern for a smaller value of generalization. Figure 5.11 shows the weight locations for a single point trained into a CMAC defined as having a base generalization of 4. Figure 5.12 shows the weight loca-

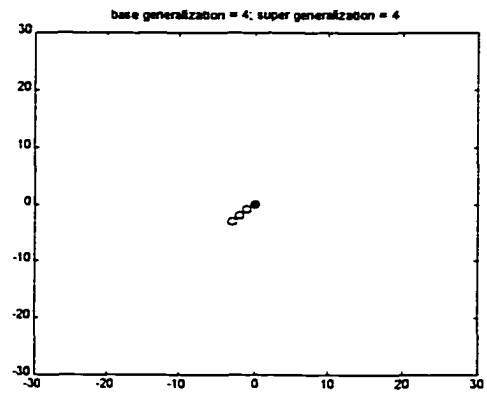


Figure 5.11: Receptive field center placement pattern for a single training point in an Albus CMAC with generalization of 4. o's represent the lower left-hand corner of a receptive field and *'s represent training points.

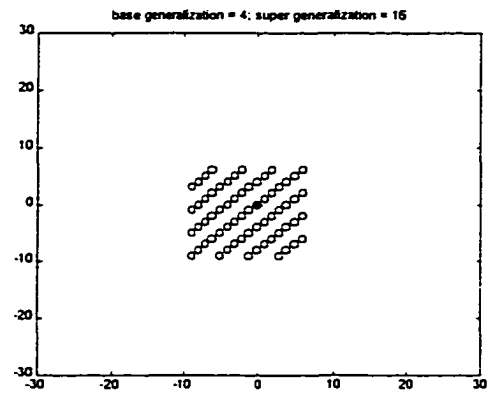


Figure 5.12: Receptive field center placement pattern for a single training point in an Albus CMAC with super generalization of 16 and base generalization of 4. o's represent the lower left-hand corner of a receptive field and *'s represent training points.

tions for the same point trained into a CMAC defined as having a base generalization of 4 but using a Super Generalization value of 16. Recall that, for the two-dimensional case, a generalization value of 16 in the standard CMAC would pack 16 weights into the same region that Super Generalization has packed 64 weights. Super Generalization has been implemented in a straightforward way in the modified UNH CMAC code and in the Matlab port of that C code. When working in the raw coordinate space of the weight space, it is true that for every weight in the weight allocation pattern such as that in figure 5.11 another weight can be found by stepping one base generalization distance in either direction along any dimension. For the case of base generalization equal to 4, once the four pri-

mary weights are located using the standard CMAC method, all the other weights in the Super Generalization region can be located by beginning at one of the primary weights and using a simple recursive function call to traverse the Super Generalization region picking up copies of the primary weight. Figures 5.13 and 5.14 show the learning support

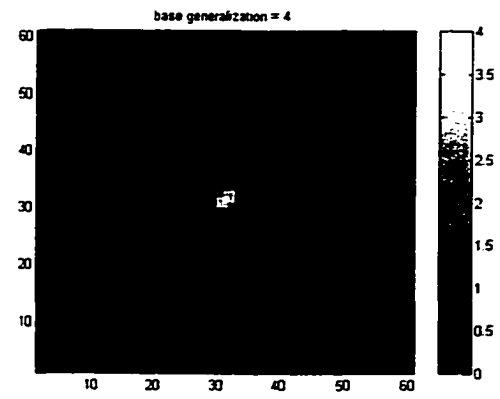


Figure 5.13: Learning support for a single training point with an Albus CMAC with generalization of 4

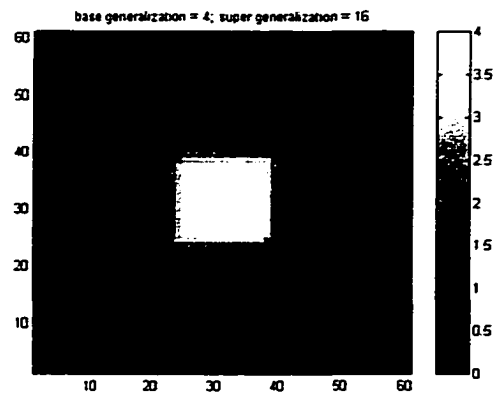


Figure 5.14: Learning support for a single training point with an Albus CMAC with super generalization of 16 and base generalization of 4.

provided by each of these methods. For the case with Super Generalization, training was done with Super Generalization while network responses were determined using the base generalization. The Super Generalization provides support to a larger region as if that region were fully trained.

5.2 Super Generalization Experiments

Several experiments were run to test the effectiveness of Super Generalization. In order to explore the effects of training sample spacing and sample spacing in general on the ability of the CMAC to learn and remember the function, the values of X and Y were scaled by an integer value before presenting the states to the CMAC for learning. Every

third sample in the domain of X and Y was used for training and then remembering was performed over the entire domain. Random selection of the training point order was not applied. The value of Super Generalization was fixed at 32 which is roughly the extent of the smoothing function as determined from the analysis of the matrix optimum solution. A new receptive field shape was defined for this problem and applied to the CMAC for learning with Super Generalization as part of the custom CMAC feature in the code. The receptive field shape is plotted below in figure 5.15. For Super Generalization regions

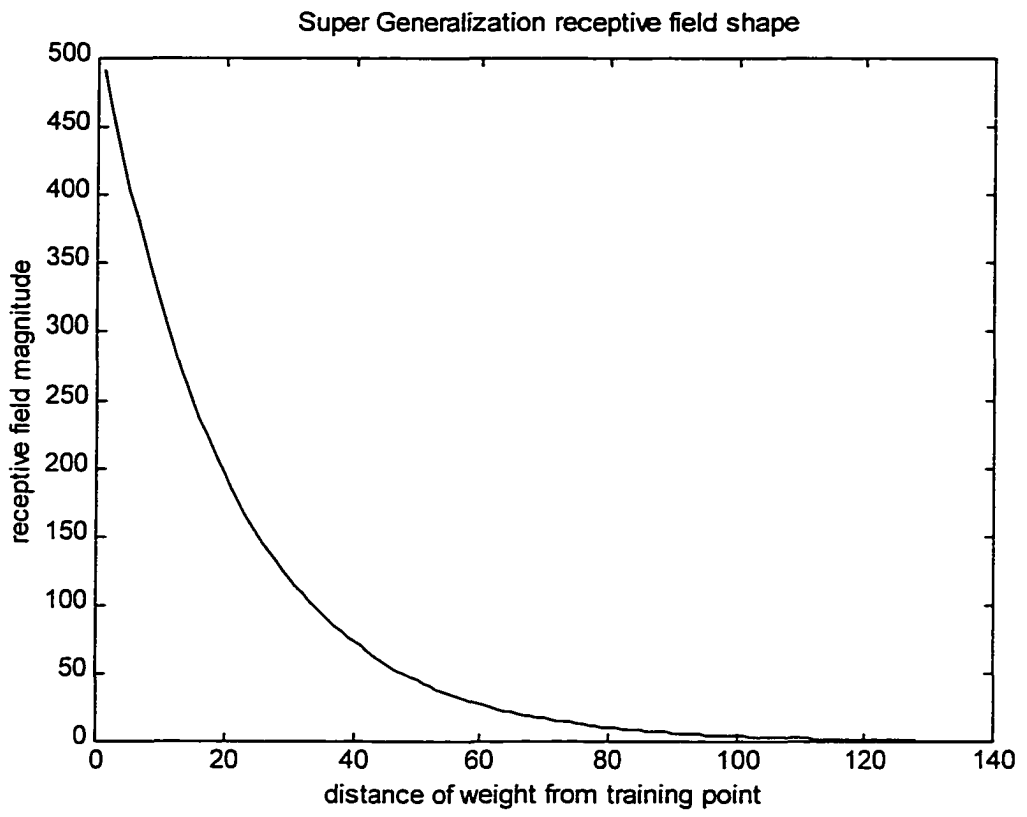


Figure 5.15: Receptive field shape for Super Generalization.

smaller than the 128 units on a side for which the function is defined, the curve is sub-sampled along its domain so that the shape is preserved.

The experiment varies the base generalization parameter from 1 to 32 while the Super Generalization parameter remains fixed at a value of 32. The CMAC learning rate is fixed at 0.5. Quantization values are fixed at 1. Available memory is 30000 and collision avoidance is active. For each value of generalization three CMACs are defined. The first is the original Albus CMAC with 20 iterations of training done across the whole training set, the second is a CMAC using only Super Generalization trained for 20 iterations, and the third is a CMAC trained for 4 iterations using Super followed by 20 iterations at the base generalization. For each value of generalization, the rms error between desired function and the learned functions for each of the three CMACs, the rms error between the X- and Y-gradients of the desired function and those of the learned functions for each of the three CMACs, and the rms error between the discrete Laplacian of the desired function and that of the learned functions for the three CMACs are calculated. All the rms errors were calculated for the full array as well as for the inside array which excluded the outer 3 rows and columns in an effort to normalize for edge effects. This same experiment was performed for three different functions.

5.2.1 Experiment 1: Up Parabola

The function being trained is defined by Eqn 5.1.

$$Z = X^2 + Y^2, \qquad X, Y \in \{-10, -9 \ldots 9, 10\} \qquad \text{Eqn 5.1}$$

The sample spacing parameter was set to a value of 4. Since every third sample was used, training samples were effectively spaced at a distance of 12 from each other in each dimension for presentation to the network. Figure 5.16 shows a plot of the full rms error for the function learning. Figure 5.17 shows a plot of the inside rms error for the function learning. These results indicate that the combination of a standard CMAC with a

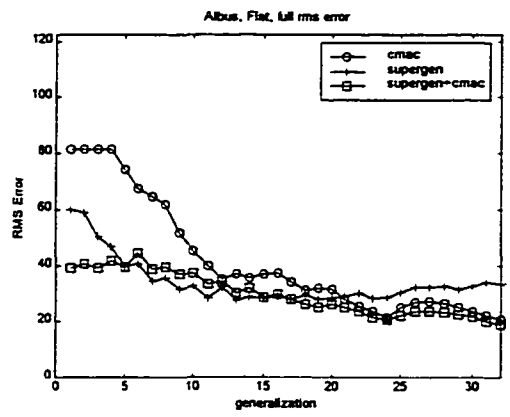


Figure 5.16: Full rms error between desired function and learned functions.

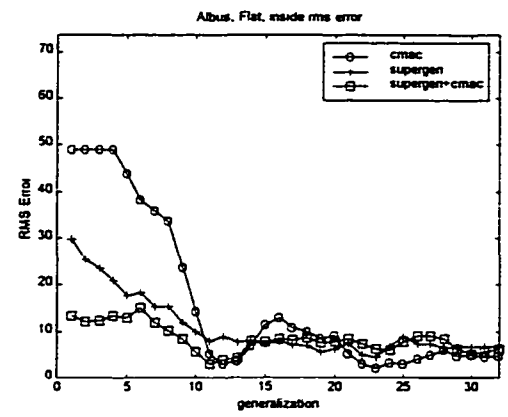


Figure 5.17: Inside rms error between desired function and learned functions.

Super Generalization CMAC has function learning performance that is superior to the standard CMAC especially at low values of the generalization parameter where Super Generalization has its most significant effect. As the value of base generalization increases, Super Generalization becomes more like standard generalization and the two CMACs converge in performance.

Figure 5.18 is a plot of the full rms error between the Laplacian of the desired function and that of the learned functions. Figure 5.19 is a plot of the inside rms error between the Laplacian of the desired function and that of the learned function. The Laplacian operator is a measure of the curvature of the function. These results indicate that the combination of regular CMAC with a Super Generalization CMAC has superior performance for small generalizations and has similar performance to the standard CMAC at high generalizations.

Figure 5.20 is a plot of the full rms error between the X-gradient of the desired function and that of the learned functions. Figure 5.21 is a plot of the inside rms error between the X-gradient of the desired function and that of the learned functions. Figure

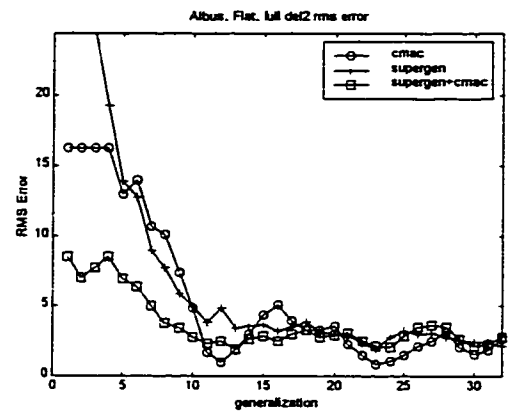


Figure 5.18: Plot of the full rms error between the Laplacian of the desired function and that of the learned function.

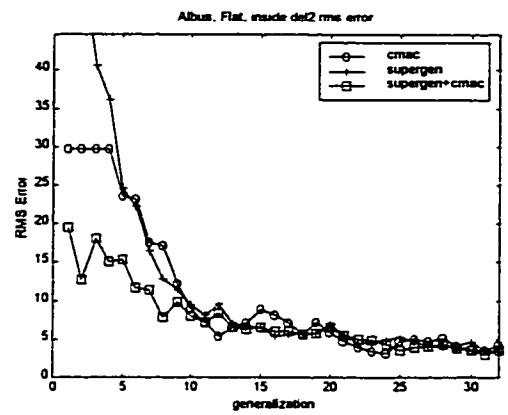


Figure 5.19: Plot of the inside rms error between the Laplacian of the desired function and that of the learned function.

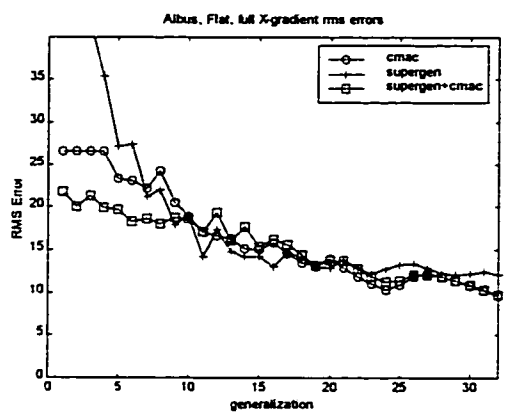


Figure 5.20: Plot of the full rms error between the X-gradient of the desired function and that of the learned functions.

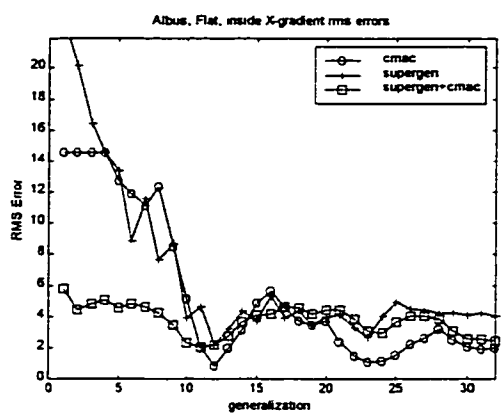


Figure 5.21: Plot of the inside rms error between the X-gradient of the desired function and that of the learned functions.

5.22 is a plot of the full rms error between the Y-gradient of the desired function and that of the learned functions. Figure 5.23 is a plot of the inside rms error between the Y-gradient of the desired function and that of the learned functions. Once again, for small values of base generalization, the combination of standard CMAC with Super Generalization CMAC produces the best derivative performance. Unlike the flawed matrix implementa-

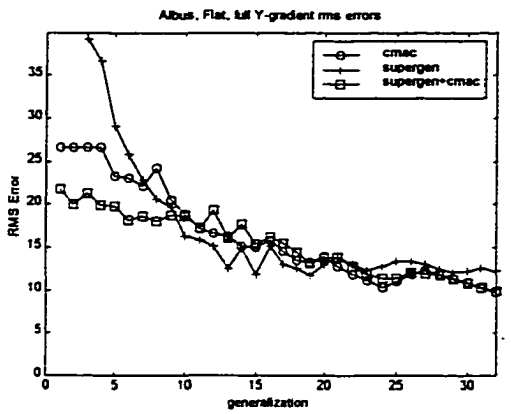


Figure 5.22: Plot of the full rms error between the Y-gradient of the desired function and that of the learned functions.

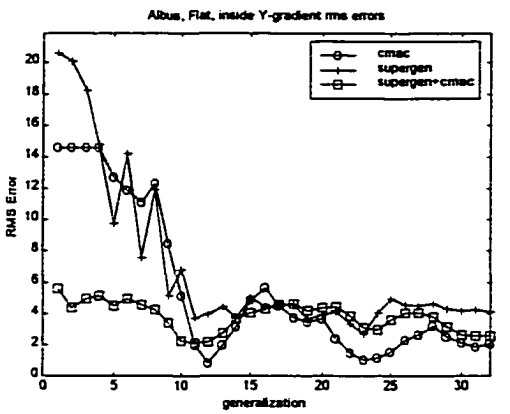


Figure 5.23: Plot of the inside rms error between the Y-gradient of the desired function and that of the learned functions.

tion of the previous chapter, this network exhibits improvement in performance for both the X- and Y-gradients. Since the weight corrections and weighting functions are calculated prior to mapping to a physical memory array, biases from this mapping process can not affect the calculations.

It is worth noting that there are large differences between the performance plots for the full array and those for the inside array. This indicates that edge effects were a big contributor to the CMAC error performance. This is not surprising since the CMAC has a natural bias toward zero. The edges of this parabola function had values on the order of 100. Since training was not guaranteed to occur at the exact edges due to the choice of every third sample, the weight information contributing to the network response at the edges is either zero if no generalization has reached that far or is information from training points on the parabola which are guaranteed to be smaller than the desired value. The next experiment further highlights this point.

5.2.2 Experiment 2: Down Parabola

The function being trained is defined by Eqn 5.2.

$$Z = 200 - (X^2 + Y^2), \qquad X, Y \in \{-10, -9 \ldots 9, 10\} \qquad \text{Eqn 5.2}$$

The sample spacing parameter was set to a value of 4. Since every third sample was used, training samples were effectively spaced at a distance of 12 from each other in each dimension for presentation to the network. Figure 5.24 shows a plot of the full rms error for the function learning. Figure 5.25 shows a plot of the inside rms error for the

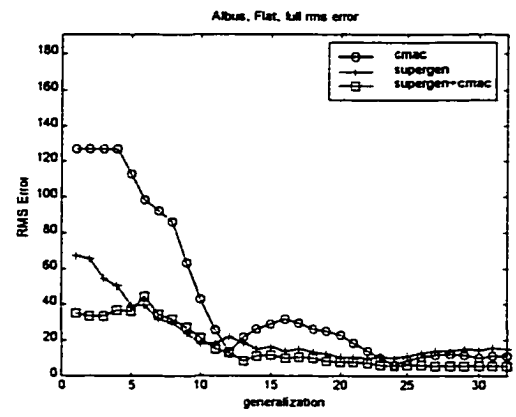


Figure 5.24: Plot of the full rms error for the function learning.

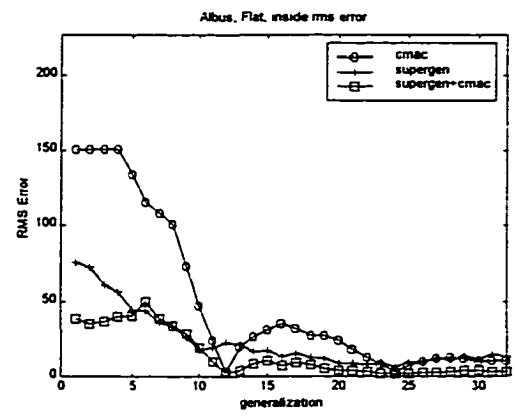


Figure 5.25: Plot of the inside rms error for the function learning.

function learning. For this function the performance of the combination of regular CMAC with the Super Generalization CMAC is better at every value of base generalization. In many cases the performance was better by a factor of two or more.

Figure 5.26 is a plot of the full rms error between the Laplacian of the desired function and that of the learned functions. Figure 5.27 is a plot of the inside rms error between the Laplacian of the desired function and that of the learned functions. For this

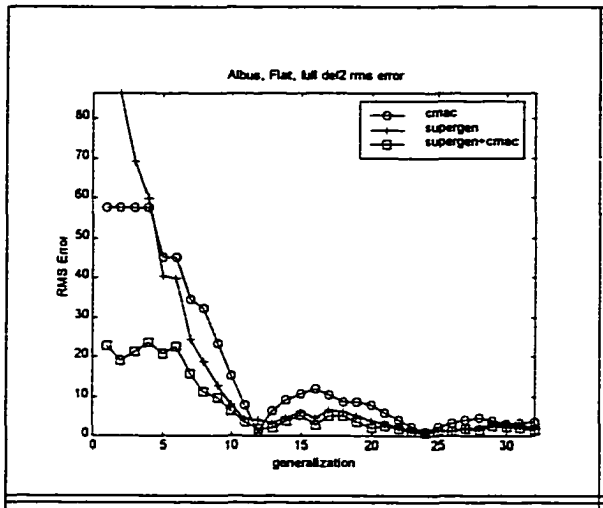


Figure 5.26: Plot of the full rms error between the Laplacian of the desired function and that of the learned functions.

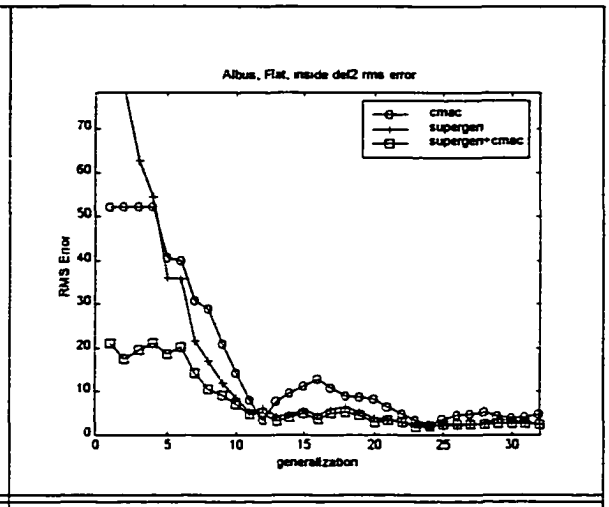


Figure 5.27: Plot of the inside rms error between the Laplacian of the desired function and that of the learned functions.

function the curvature performance of the combination of regular CMAC with the Super Generalization CMAC is better at every value of base generalization. In all but four cases the performance was better by a factor of 1.5 or better.

Figure 5.28 is a plot of the full rms error between the X-gradient of the desired function and that of the learned functions. Figure 5.29 is a plot of the inside rms error

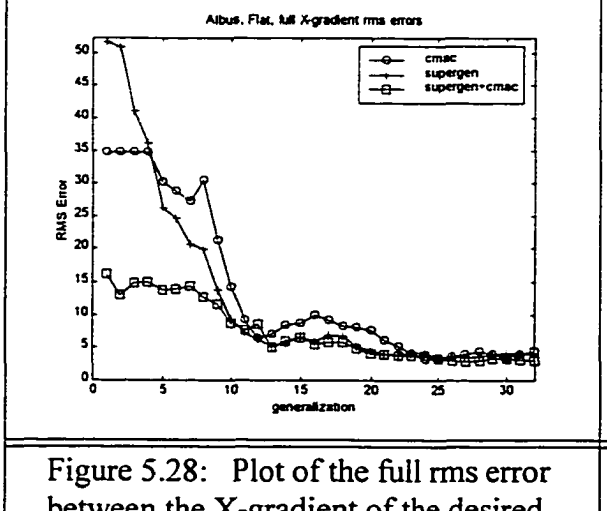


Figure 5.28: Plot of the full rms error between the X-gradient of the desired function and that of the learned functions.

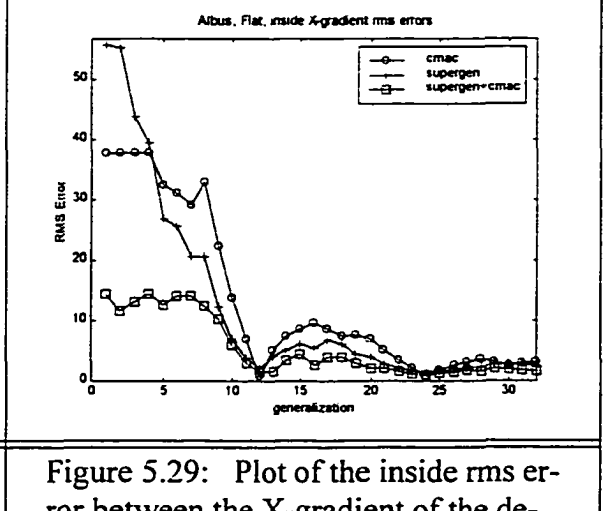


Figure 5.29: Plot of the inside rms error between the X-gradient of the desired function and that of the learned functions.

between the X-gradient of the desired function and that of the learned functions. Figure 5.30 is a plot of the full rms error between the Y-gradient of the desired function and that of the learned functions. Figure 5.31 is a plot of the inside rms error between the Y-gradi-

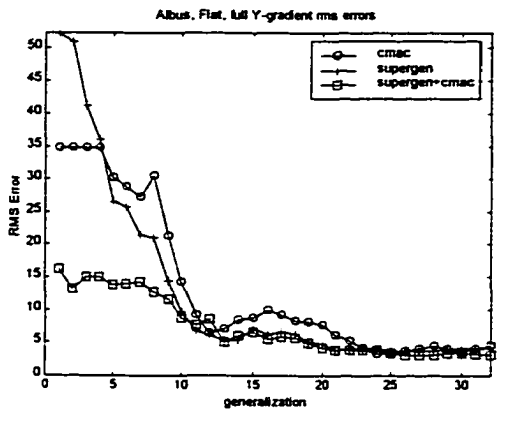


Figure 5.30: Plot of the full rms error between the Y-gradient of the desired function and that of the learned functions.

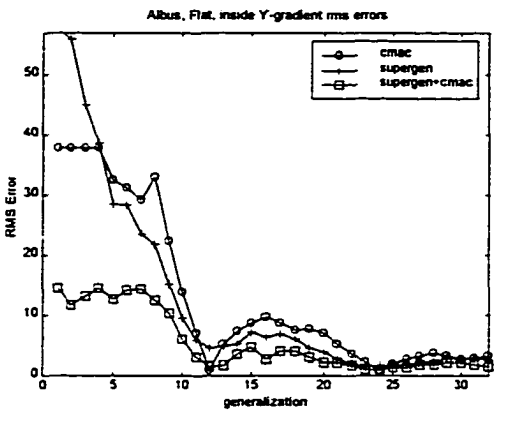


Figure 5.31: Plot of the inside rms error between the Y-gradient of the desired function and that of the learned functions.

ent of the desired function and that of the learned functions. For all but three cases, the derivative performance in both the X- and Y-directions of the combined standard CMAC plus Super Generalization CMAC was superior to the standard CMAC alone.

It is interesting to note that for this function, which has edge values near zero, the full and insides plots are very similar. This indicates that edge effects were not a significant contributor to the errors

5.2.3 Experiment 3: Sin(X)*Sin(Y)

The function being trained is defined by Eqn 5.3.

$$Z = 100 \sin X \cdot \sin Y, \qquad X, Y \in \left\{ -\pi : \frac{2\pi}{50} : \pi \right\} \qquad \text{Eqn 5.3}$$

The sample spacing parameter was set to a value of 20. Since every third sample was used, training samples were effectively spaced at a distance of about 7.5 from each other in each dimension for presentation to the network.

Figure 5.32 shows a plot of the full rms error for the function learning. Figure 5.33

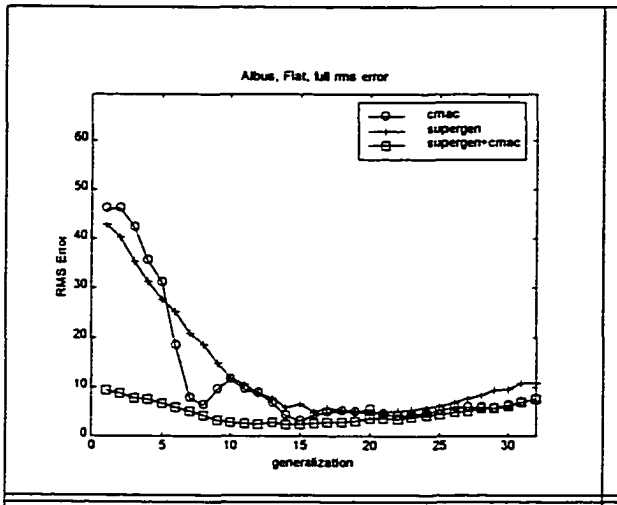


Figure 5.32: Plot of the full rms error for the function learning.

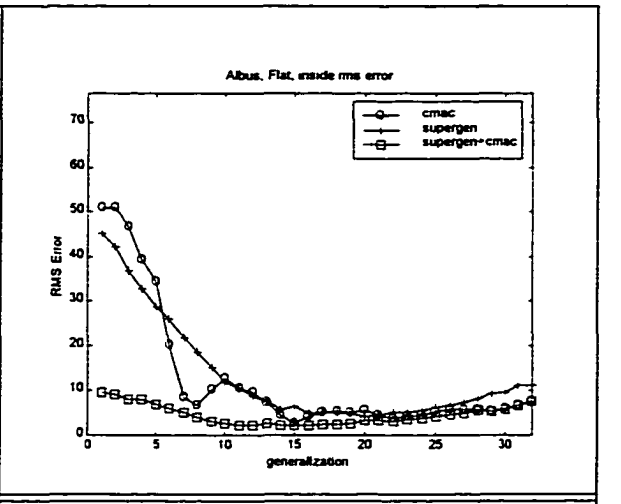


Figure 5.33: Plot of the inside rms error for the function learning.

shows a plot of the inside rms error for the function learning. The performance of the combined regular plus Super Generalization CMAC is better than or equal to that of the regular CMAC alone. For the high values of base generalization, where Super Generalization behaves almost exactly like the base generalization, function learning performance is identical. For small values of base generalization, function learning performance for the combined CMAC is up to five times better.

Figure 5.34 is a plot of the full rms error between the Laplacian of the desired function and that of the learned functions. Figure 5.35 is a plot of the inside rms error between the Laplacian of the desired function and that of the learned functions. In all but five cases the performance of the combined CMAC was better than the regular CMAC for

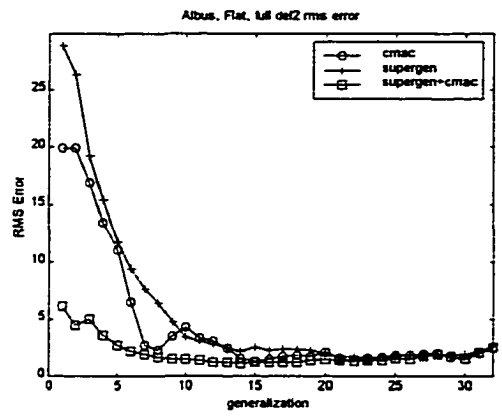


Figure 5.34: Plot of the full rms error between the Laplacian of the desired function and that of the learned functions.

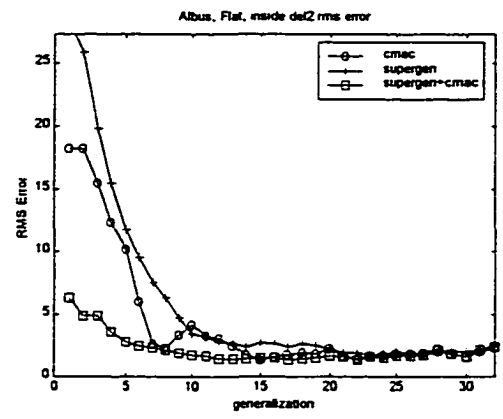


Figure 5.35: Plot of the inside rms error between the Laplacian of the desired function and that of the learned functions.

this performance metric. In no case was the performance of the combined CMAC less than 90 percent of the regular CMAC.

Figure 5.36 is a plot of the full rms error between the X-gradient of the desired function and that of the learned functions. Figure 5.37 is a plot of the inside rms error

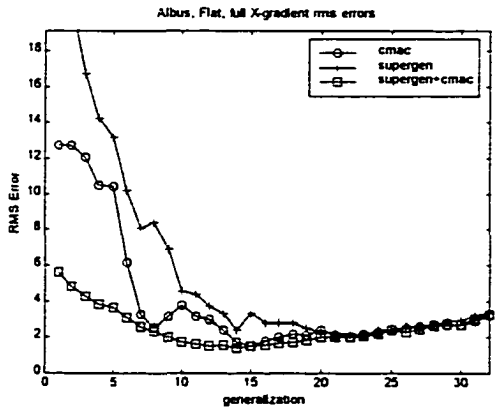


Figure 5.36: plot of the full rms error between the X-gradient of the desired function and that of the learned functions.

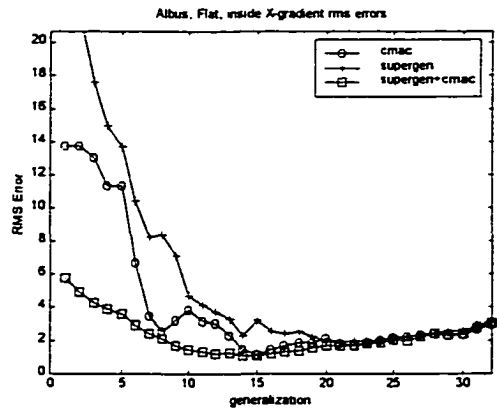


Figure 5.37: Plot of the inside rms error between the X-gradient of the desired function and that of the learned functions.

between the X-gradient of the desired function and that of the learned functions. Figure 5.38 is a plot of the full rms error between the Y-gradient of the desired function and that of the learned functions. Figure 5.39 is a plot of the inside rms error between the Y-gradi-

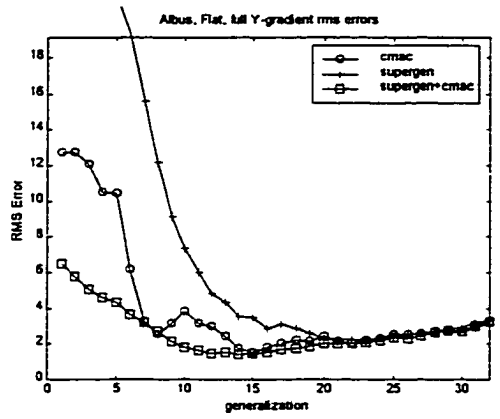


Figure 5.38: Plot of the full rms error between the Y-gradient of the desired function and that of the learned functions.

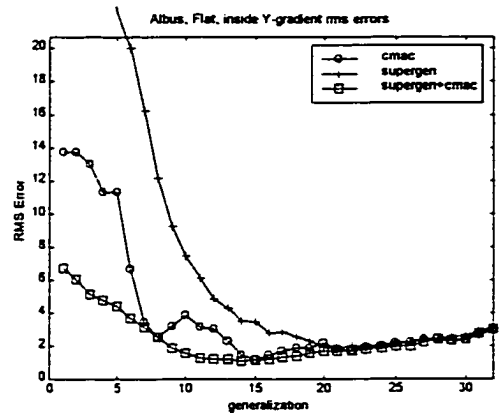


Figure 5.39: Plot of the inside rms error between the Y-gradient of the desired function and that of the learned functions.

ent of the desired function and that of the learned functions. In no case was the derivative performance of the combined CMAC less than 93 percent of the regular CMAC and in most cases the performance was much better. Again, the similarity between the full and inside plots indicates that edge effects were not a factor in this experiment as would be expected for a function whose value is zero at the boundary of the training region.

Results with the Super Generalization CMAC were disappointing but the poor performance arose because the receptive field normalization was calculated using the existing tapered receptive field algorithm. This algorithm normalizes the individual weight contributions based upon the assumption that all the weights in the receptive field will be used to produce the network response. The effect of this was particularly catastrophic when the

values of the base generalization were very small. When the base generalization is very small there can be up to 1024 active weights within the Super Generalization region. Several of these weights must lie very close to the center of the Super Generalization receptive field and their weight values are assigned correspondingly high contributions of the input data point. For tapered receptive fields, the field normalization is applied both during training and during remembering. When remembering with a small receptive field near the center of a Super Generalization cell, the normalization on the remember side is completely inadequate to compensate for the huge contributions stored in those weights during the training cycle and the resulting network response is excessively large resulting in a large rms error near these locations and a very spiky network response. When remembering with the Albus law, the weights are simply summed to produce the network response resulting in even larger errors.

This effect was virtually eliminated by simply retraining the network with the standard Albus CMAC law after some initial training with Super Generalization. This served to renormalize the central set of weights in the Super Generalization cell and dramatically improved the overall network performance.

The overall results indicate that a CMAC trained with properly normalized Super Generalization can have superior performance over the standard CMAC when the training is sparse or incomplete. As the value of base generalization increases the performance of the new CMAC approaches that of the standard CMAC although even at large values of base generalization the performance of the new CMAC was often still marginally better than the standard CMAC.

Chapter 6

Conclusions and Suggestions for Future Work

6.1 Summary

The stability of the CMAC algorithm was analyzed and a Lyapunov stability proof was developed for the open loop, function learning case. The results indicated that as long as the function could be represented to the desired accuracy by a fully trained CMAC, then the weights in the CMAC being trained would eventually converge to the weights of the fictitious target CMAC. The Lyapunov function was defined on the error between the weights of the CMAC being trained and the weights of the target CMAC. This weight error was shown never to increase and, in fact, for the weights being trained was monotonically decreasing.

This matrix formulation of the CMAC was also used to cast the learning algorithm as a constrained optimization problem. Lagrange multipliers were used and both batch and sequential updating were studied for the one- and two- dimensional problems. As much as possible of the CMAC structure was included in the matrix formulation. This was especially important for the two-dimensional case where both the Albus receptive field center placement strategy and the An receptive field placement strategy were supported. Tapered receptive fields were also supported in this matrix formulation with code from the original CMAC C code ported for the C to Matlab. Floating point weights were used which

allowed for a larger dynamic range for learning in the matrix case than in the standard C-code implementation. In practice this was not significant. The purpose of this analysis was to explore weight smoothing or regularization from an optimization standpoint and determine if learning performance was improved. If this was the case, then important features of the optimum solution would be extracted and cast in a form suitable for the CMAC.

An important finding from this matrix analysis was that the point in the process of mapping from an n-dimensional input/weight space to a one-dimensional weight vector at which the weight adjustment occurred had a significant impact on performance. In all the matrix experiments the formulation required that operations be performed on the one-dimensional weight vector. Some operations, like receptive field center placement, were performed in the higher dimensional space but all results were mapped to the linear space before the matrix algorithm was calculated. Even the weight penalty matrix developed for smoothing only smoothed on nearest neighbors in the weight array and not necessarily on nearest neighbors in the original weight space. This resulted in the documented bias to smoothing along a preferred dimension as shown by the first derivative tests on solutions from the matrix optimizations.

Despite the apparent flaw in the implementation, weight smoothing resulted in significant improvements in learning performance. Three metrics in particular were deemed important: the network performance under partial training, the first derivative of the network solution, and the smoothness, measured by the laplacian, of the network solution Performance graphs against these three metrics were presented in the previous chapter. Important properties of the weight smoothing solution derived from the optimum solutions are listed here.

1) Application of weight smoothing as derived from the solutions to the constrained optimization problems resulted in a significant improvement in network performance particularly at the smaller values of generalizations where the extended generalization effect of the weight smoothing was most strongly felt.

2) Application of weight smoothing as derived from the solutions to the constrained optimization problems requires the introduction of a new feature to the CMAC -- Super Generalization. Whereas standard generalization in the CMAC defines a hypercube with dimension C on each side within which C receptive fields overlap, Super Generalization defines a hypercube of integer dimension G_s on a side with $G_s > C$. The distribution of receptive field centers in this larger region is the same as that defined by CMAC for generalization of C. In this way all the information trained into the CMAC will be available for network responses at nearby locations in the input space.

3) The kernel function for the weight smoothing solution is derived from the inverse of the penalty matrix $\underline{Q}$ as defined in Eqn 4.7 and on average takes the form

$$\frac{1.5617}{1.36426^{|x - x_0|}} \qquad \text{Eqn 6.1}$$

This is the desired shape for training using Super Generalization although some changes to the actual parameters may be required to accommodate the present implementation strategy in the UNH CMAC for integer performance and tapered receptive field shapes.

4) The weight smoothing solution applies the kernel function, convolved with the weight selection vector, as a weighting factor for the new training point. This new weighting factor spreads the information out much farther into the weight space than is possible

with all but the largest values of generalization. The significant feature is that the width of the spreading is controlled by the scaling of the penalty matrix and this information is applied to the weights in a manner that is independent of the base value of generalization for which the network was created.

5) The weight smoothing solution applies the kernel function to average the old weights before a new sample value is trained. This process, defined by term 1 (Eqn 4.18), enforces weight smoothing across the old weights and also acts to control the problem of runaway weights in the standard CMAC implementation. The requirement placed on the CMAC algorithm is that, for every training iteration, a pre-training step must be added which performs this weight averaging process across the accessible weights within the range of the smoothing kernel.

6) The weight smoothing solution determines a measure of the interaction between the new training point and previously trained data and adjusts both the weight averaging function and the new sample training function in such a way as to minimize interference with previously trained information while still enforcing the smoothness criterion.

The UNH CMAC code was modified to support the kernel requirement of property 4 above with the implementation of Super Generalization. This implementation evolved from a study of how the CMAC algorithm actually does generalization, how different values of generalization select sets of weights from the global weight space, and how these sets of weights interact under mixed generalization training and remembering. Experiments were run with the new version of the CMAC algorithm.

6.2 Conclusions

Experiments were conducted with three different CMAC networks. The first was the standard Albus CMAC algorithm, the second was a CMAC where training was performed with Super Generalization and remembering was performed with the standard Albus algorithm, and the third was a CMAC where the first few training passes were performed with Super Generalization and the remaining training passes were performed with the standard Albus algorithm. Remembering for the third network was also performed with the standard Albus algorithm.

Results with the Super Generalization CMAC were disappointing but the poor performance arose because the receptive field normalization was calculated using the existing tapered receptive field algorithm. This algorithm normalizes the individual weight contributions based upon the assumption that all the weights in the receptive field will be used to produce the network response. The effect of this was particularly catastrophic when the values of the base generalization were very small. When the base generalization is very small there can be up to 1024 active weights within the Super Generalization region. Several of these weights must lie very close to the center of the Super Generalization receptive field and their weight values are assigned correspondingly high contributions of the input data point. For tapered receptive fields, the field normalization is applied both during training and during remembering. When remembering with a small receptive field near the center of a Super Generalization cell, the normalization on the remember side is completely inadequate to compensate for the huge contributions stored in those weights during the training cycle and the resulting network response is excessively large resulting in a large rms error near these locations and a very spikey network response. When remembering with the Albus law, the weights are simply summed to produce the network

response resulting in even larger errors.

This effect was virtually eliminated by simply retraining the network with the standard Albus CMAC law after some initial training with Super Generalization. This served to renormalize the central set of weights in the Super Generalization cell and dramatically improved the overall network performance.

The overall results indicate that a CMAC trained with properly normalized Super Generalization can have superior performance over the standard CMAC when the training is sparse or incomplete. As the value of base generalization increases the performance of the new CMAC approaches that of the standard CMAC although even at large values of base generalization the performance of the new CMAC was often still marginally better than the standard CMAC.

The research performed for this dissertation indicates that, with a few more refinements, this new CMAC learning algorithm will make it possible to use CMAC in applications for which it is now unsuited such as optimization problems. This is a particularly interesting result in that the training with Super Generalization does not, in and of itself, implement weight smoothing or even any obvious approximation to weight smoothing. All Super Generalization training does is spread the trained information farther out into the weight space in a form that makes the network response much smoother for cases when the required network base generalization is small and/or the training data in all or portions of the state space are widely separated. Nevertheless, this result is significant since it has a low computational cost for modest problems. The computational scale factor is proportional to:

$$\left(\frac{\text{super generalization parameter value}}{\text{base generalization parameter value}}\right)^{\text{Dimension}}$$

Despite the now exponential cost growth with dimension, this learning rule is still a local update learning rule and as such still maintains those traditional advantages over the global updating networks. Even if the entire weight smoothing rule were to be implemented, the algorithm would still be a local update law although the cost associated with all the weight averaging and smoothing would be proportional to:

$$(\text{super generalization parameter value})^{\text{Dimension}}$$

In the final vision of how this new learning technique might be used, it is expected that Super Generalization and weight smoothing will be used only occasionally and not necessarily together. For those times when training is sparse and the differentiability of the network response is required then both will probably prove useful to maximize performance. Training interaction detection will be implemented so that, in conjunction with variable Super Generalization cell size, the compensation mechanism of the optimal solution can be approximated in the CMAC as well to minimize learning interference. For the case when data density is good, perhaps Super Generalization will not be necessary at all. In this case, however, it may still prove useful to perform weight smoothing using a scaled version of the smoothing kernel over some region closer in size to the base generalization just to implement local smoothing and weight magnitude control. In intermediate regions the training interaction detector can signal whether training data are sparse enough to warrant the application of Super Generalization just for a single sample or short series of training data in order to create a reasonable base from which to draw network responses in future visits to that region of the input space.

6.3 Suggestions for Future Work

1) Since the Super Generalization technique appears to be viable, the first thing that

should be done is to develop a Super Generalization receptive field normalization, based on the existing tapered receptive field technique, that the information stored in the weights will be scaled so that recovery of that information with the base generalization will result in correctly scaled responses. This will eliminate the need to post train at the training points with an update law operating at the base generalization as was done in the experiments presented here.

2) A full-blown implementation of weight smoothing for the old weights should be implemented and tested now that the optimal weight smoothing function has been derived from the optimal matrix implementation. While computationally more expensive than other CMAC sub-algorithms, this could still be implemented with only table lookup and the basic arithmetic operations. The new computer hardware should be able to support the extra computation and still allow for good real time performance. Even a dedicated hardware implementation in Field Programmable Gate Array (FPGA) technology or Application Specific Integrated Circuitry (ASIC) should be possible.

3) One of the big strengths of the optimum matrix solution is the way the algorithm detects when there will be interaction between the new training point and existing trained weights and prescales both the smoothed old weights and the Super Generalized new data point to minimize the disruption and in fact to improve the smoothness of the overall solution. This should be implemented even if weight smoothing is not implemented. Now that a form of variable generalization is available since Super Generalization Cell size can be adjusted dynamically without disrupting the underlying base generalization structure, it makes sense to dynamically adjust the cell size to get

optimum coverage and to respond to changing density of training data.

4) If possible the matrix formulation used in the two-dimensional implementation of the optimization problem should be corrected to accurately implement smoothing in both the X- and Y-directions. If this can be accomplished and an appropriate penalty matrix can be constructed, then new insights into the properties of the weight smoothing algorithm might present themselves.

5) The new weight adjustment law or some variation of it should be recast in matrix form so that the local generalization property is built into the algorithm in an explicit way, thereby differentiating it from the global form represented by the matrix optimum solution. This local update law should be analyzed for stability.

References and Bibliography

[1] Albus, J. S., "A Theory of Cerebellar Functions," Mathematical Biosciences 10, pp. 25-61, 1971.

[2] Albus, J. S., "Theoretical and Experimental Aspects of a Cerebellar Model." Ph.D. Dissertation, University of Maryland, 1972.

[3] Albus, J. S., "The Cerebellar Model Articulation Controller, Trans. ASME. Series G, vol. 97, No. 3, 1975.

[4] Albus, J. S., "A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC)." Trans. ASME, J. Dynamic Syst. Meas. Contr., Transactions of ASME, vol 97, pp. 220-227, 1975. (September 1975)

[5] Albus, J. S., "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," Journal of Dynamic Systems, Measurement, and Control, Transactions of ASME, PP. 228-233, September 1975.

[6] Albus, J. S., "A Model of the Brain for Robot Control Part3: A comparison of the Brain and Our Model," Byte Magazine, 1979.

[7] Albus, J. S., "Mechanisms of Planning and Problem Solving in the Brain," Mathematical Biosciences, Vol. 45, pp. pp. 247-293, 1979.

[8] Albus, J. S., *Brains, Behavior and Robotics*, BYTE Publications Inc., Peterborough, New Hampshire, 1981.

[9] An, P.-C. E., "An Improved Multi-Dimensional CMAC Neural Network: Receptive Field Function and Placement." Ph.D. Dissertation, Univ. of New Hampshire, Durham, NH, September, 1991.

[10] An, P.-C., E., Miller, W. T., and Parks, P. C., "Design Improvements in Associative Memories for CMAC." Proc. ICANN '91, vol. 2., pp. 1207-1210, (North Holland Pub.), Helsinki, June 24-28, 1991.

[11] Anagnost, S. M., and Glanz, F. H., "CMAC Neural Network Binary Output Capacity." UNH Intelligent Structures Group Technical Report No. ECE.IS.91.01, ECE Dept., Univ. of New Hampshire, Durham, NH, 1991.

[12] Anagnost, S. M., "Approximate non-linear optimal control using CMAC neural networks," Masters Thesis, University of New Hampshire, 1994.

[13] Anderson, J. A., Silverstein, J. W., Rite, S. A., and Jones, R. S., "Distinctive Features, Categorical Perception, and Probability Learning: some Applications of a Neural Model," Psych. Rev. 84 413-451, 1977.

[14] Arehart, K. F., "A CMAC-based cursive handwriting recognizer for the Windows for Pen Computing operating environment," Masters Thesis, University of New Hampshire, 1994.

[15] Bergantz, D. and Barad, H., "Neural network control of cybernetic limb prostheses," Annual International Conference of the IEEE Engineering in Medicine and Biology Society, New Orleans, LA, Vol. 3, pp. 1486- 1487, 1988.

[16] Botros, S. M., and Atkeson, C. G., "Generalization Properties of Radial Basis Func-

tions." In *Neural Information Processing Systems 3*, edited by R. P. Lippmann, John E. Moody, and David S. Touretzky, Morgan Kaufmann Publishers, San Mateo, CA, pp. 707-713, 1991.

[17] Briggs, R., and Glanz, F. H., "Parameter Influence on a CMAC Neural Network." UNH Intelligent Structures Group Technical Report No. ECE.IS.91.02, ECE Dept., Univ. of New Hampshire, Durham, NH, 1991.

[18] Brogan, W. L., **Modern Control Theory, third edition**, Prentice Hall, Englewood Cliffs, New Jersey, 1991, p. 131.

[19] Brown, M, "Neurofuzzy Adaptive Modelling and Control," Ph.D. Thesis, Southampton University, 1993.

[20] Brown, M., Harris, C. J. and Parks, P. C., "The Interpolation Capabilities of the Binary CMAC," Neural Networks, Vol. 6, No. 3, pp. 429-440, 1993.

[21] Brown, M, and Harris, C. J., "Comments on 'Learning convergence in the Cerebellar Model Articulation Controller,'" IEEE Trans. on Neural Networks

[22] Brown, M., Harris, C. J., and Parks, P. C., "The Interpolation Capabilities of the Binary CMAC," Neural Networks, v6, no 3, p. 429, 1993

[23] Brown, M.and Harris, C. J., "The modelling abilities of the binary CMAC," IEEE international Conference on Neural Networks, Orlando, Florida, Vol. 3, pp. 1335-1339, 1994.

[24] Brown, M. and Harris, C. J., Neurofuzzy Adaptive Modelling and Control. Hemel Hempstead, UK: Prentice Hall, 1994.

[25] Burgin, G, "Using Cerebellar Arithmetic Computers," AI Expert, vol 7, no 6, p 32, June 1, 1992.

[26] Calcev, G., "Self-tuning neurofuzzy controller," IEEE International Symposium on Intelligent Control, Chicago, IL, pp. 577-580, 1993.

[27] Carlson, R., Lee, C., and Rothermel, K., "Real time neural control of an active structure," International Conference on Artificial Neural Networks in Engineering, pp. 623-628, 1992.

[28] Carter, M. J., Rudolph, F., and Nucci, A., "Operational Fault Tolerance of CMAC Networks," appears in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (Ed.), San Mateo, CA: Morgan Kaufmann,1990

[29] Chapeau-Blondeau, F., Chauvet, G, "A Neural Network Model of the Cerebellar Cortex for Performing DynamicAssociations," Biological Cybernetics, vol 65, no 4, p 267, 1991.

[30] Cotter, N. E. and Guillerm, T. J., "The CMAC and a Theorem of Kolmogorov," Neural Networks. 5, 221-228, 1992.

[31] Cotter, N. E. and Mian, O. N., "A pulsed neural network capable of universal approximation," IEEE Transactions on Neural Networks, Vol. 3, pp. 308-314, 1992.

[32] Daarla and Zhao, "A learning algorithm for a CMAC-based system and its application to classification of ultrasonic signals," Ultrasonics, Vol. 32, pp. 91-98, 1994.

[33] Eldracher, M., Staller, A., and Pompl, R., "Function Approximation with Continuous-Valued Activation Functions in CMAC," acquired via ftp from Munich Technical University.

[34] Ellison, D., "On the Convergence of the Albus Perceptron," IMA Journal of Math. Control and Info., Vol. 5, pp. 315-331, 1988.

[35] Ellison, D., "On the Convergence of the Multidimensional Albus Perceptron," International Journal of Robotics Research, 10(4), pp. 338-357, 1991.

[36] Ersu, E., "A Learning Mechanism for an Associative Storage System," Proc. of IEEE International Conference on Cybernetics and Society, pp. 26-28, Oct., 1981.

[37] Ersu, E., "On the Application of Associative Neural Network Models to Technical Control Problems," Localization and Orientation in Biology and Engineering, Varju/Schnitzler Eds., Springer Verlag, 1983

[38] Ersu, E. Mao, X., "Control of pH Using a Self-Organizing Control Concept with Associative Memories," Int. IASTED Conf. on Applied Control and Identification, Copenhagen, Denmark, 1983.

[39] Ersu, E., "Real-Time Implementation of an Associative Memory Based Learning Control Scheme for Non-linear Multivariable Processes," 1st Measurements and Control Sysmpsium on Application of Multivariable Systems Techniques, pp. 109-119, Plymouth, UK, 1984

[40] Ersu, E.eds., On the application of associative neural network models to technical control problems, Springer Verlag, 1984, pp. 90-93.

[41] Ersu, E. and Militzer, J., "Real-time implementation of an associative memory-based learning control scheme for non-linear multivariable processes," 1st Measurements and Control Symposium on Applications of Multivariable Systems Techniques, Plymouth, UK, pp. 109-119, 1984.

[42] Ersu, E. and Tolle, H., "Learning Control Structures with Neuron-Like Associative Memory Systems," in Organization of Neural Networks, W. von Seelen, G. Shaw, U. M. Leinhos, Eds, VCH Verlagsgesellschaft mbJ, Weinheim, FRG, pp. 417-438, 1988.

[43] Ersu, E. and Tolle, H., "A New Concept for Learning Control Inspired by Brain Theory," Proc. of IFAC World Congress, Budapest, Hungary, July 2-6, 1984.

[44] Ersu, E. and Tolle, H., "Heirarchical Learning Control--An Approach with Neuron-Like Associative Memories," IEEE Conf. on Neural Information Processing Systems -- Natural and Synthetic, Nov. 8-12, Denver, CO, 1988.

[45] Eskandarian, A., Bedewi, N. E., Kramer, B., and Barbera, A. J., "Dynamics modeling of robotic manipulators using an artificial neural network," Journal of Robotic Systems, Vol. 11, pp. 41-56, 1994.

[46] Freedman, J. J., "Adaptive Control of Groundwater Flow Using the CMAC Neural Network," Masters Thesis, University of New Hampshire, 1996.

[47] Fukushima, K. and Miyaka, S. "Neocognitron: A Self-Organizing Neural network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," Biol, Cybern. 36(4), 193-202, 1980.

[48] Fukuda, T. , Saito, F., and Arai, F., "Study on the brachiation type of mobile robot (Heuristic creation of driving input and control using CMAC)," 12th. International Conference on Soil Mechanics and Foundation Engineering, Rio de Janeiro, Br., Vol. 2, pp. 478-483, 1989.

[49] Gehlen, S., Hormel, M., and Bohrer, S., "A learning control scheme with neuron-like associative memories for the control of biotechnological processes," neural networks, Nimes, France, pp. 1988.

[50] Gehlen, S. and Kreuzig, J., "Learning by interpolating memories for modelling of fermentation processes," Advanced Control of Chemical '91, Toulouse, France, pp.

273-278, 1991.

[51] Geng, Z. and Haynes, L., "Neural network solution for the forward kinematics problem of a Stewart platform," 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, Vol. 3, pp. 2650-2655, 1991.

[52] Geng, Z. and Haynes, L. S., "Dynamic control of a parallel link manipulator using CMAC neural network," IEEE International Symposium on Intelligent Control, Arlington, VA, pp. 411-416, 1991.

[53] Geng, Z. and Haynes, L. S., "Neural network solution for the forward kinematics problem of a Stewart platform," Robotics and Computer-Integrated Manufacturing, Vol. 9, pp. 485-495, 1992.

[54] Geng, Z and Haynes, L. S., "Dynamic Control of a Parallel Link Manipulator using a CMAC Neural network," Computers and Electrical Engineering, vol 19, no 4, p. 265, July, 1, 1993.

[55] Glanz, F. H., Miller, W. T., "Shape recognition using a CMAC based learning system," Proceedings SPIE: Intelligent Robots and Computer Vision, Cambridge, Mass., Nov., 1987.

[56] Glanz, F. H., and Miller, W. T., "Deconvolution and Nonlinear Inverse Filtering Using a Neural Network," International Conference on Acoustics and Signal Processing, Glasgow, Scotland, Vol. 4, pp. 2349-2352, 1989

[57] Glanz, F. H., and Miller, W. T., "Deconvolution using a CMAC neural network," First Annual Conference of the International Neural Network Society, Boston, MA, September 6-10, 1988, p. 440.

[58] Glanz, F. H., and Yang, J., "Experimental parameter studies for the CMAC neural network," IJCNN-91, Seattle, WA, p. 975, 1991.

[59] Glanz, F.H, Miller, W.T., and Kraft, L.G., "An overview of the CMAC neural network," IEEE Conference on Neural Networks for Ocean Engineering, Washington, DC, pp. 301-308, 1991.

[60] Grossberg, S., "Classical and Instrumental Learning by Neural Networks," in *Progress in Theoretical Biology*, New York, Academic Press, vol 3, pp 51-141, 1977.

[61] Hagens, A, and Doveton, J. H., "Application of a Simple Cerebellar Model to Geologic Surface Mapping," Computers and Geosciences, vol. 17, no 4, p 561, 1991.

[62] Hebb, D. O. *The Organization of Behavior, A Neuropsychological Theory*, New York, John Wiley, 1949.

[63] Herold, D. J., Miller, W. T., Kraft, L. G., Glanz, F. H., "Pattern Recognition using a CMAC Based Learning System," Proceedings SPIE: Automated Inspection and High Speed Vision Architectures II, Vol. 1004, pp. 84-90, 1988.

[64] Hewes, R. P., and Miller, W. T., "Practical demonstration of a learning control system for a five axis industrial robot," Proceedings SPIE: Intelligent Robots and Computer Vision, vol. 1002, pp. 679-685, 1988.

[65] Jager, R., "Fuzzy Logic in Control," Ph.D. Dissertation, Delft University of Technology, The Netherlands, 1995.

[66] Jia, Y, "Use of the Sphere Packing Lattice to Investigate the Quality of CMAC Receptive Field Placement," Masters Thesis, University of New Hampshire, 1994.

[67] Jin, Y., Pipe, T., and Winfield, A., "Stable neural network control for manipulators," International Joint Conference on Neural Networks, Nagoya, Jpn, Vol. 3, pp.

2775-2778, 1993.

[68] Kano, H. and Takayama, K., "Learning Control of Robotic Manipulators Based on Neurological Model CMAC," Proceedings of the 11th Triennial World Congress of the International Federation of Automatic Control, Tallinn, USSR, p. 249-254, Aug. 13-17, 1990.

[69] Kim, H, "CMAC-Based Adaptive Critic Self-Learning Control," IEEE Trans. on Neural Networks, vol. 2, no 5, p 530, 1991.

[70] Kohonen, T., *Associative Memory: A System-Theoretical Approach*, Berlin, Springer Verlag, 1977.

[71] Kolez, A. and Allinson, N. M., "Realisation of a modified CMAC architecture using reconfigurable logic devices." 3rd Workshop on Neural Networks: Academic/Industrial/NASA/Defense, Auburn, AL, SPIE Vol. 1721, pp. 195-206, 1993.

[72] Kraft, L. G., and Campagna, D. P., "A Comparison of CMAC Neural Network and Traditional Adaptive Control Systems." Proc. of the 1989 American Controls Conf., Pittsburgh, Pa., May, 1989

[73] Kraft, L.G., An, Edgar, and Briggs, E., "Convergence Properties of CMAC Neural Network Controllers", Proceedings of the 1991 American Controls Conference, Boston, Mass, June, 1991.

[74] Kraft, L. G., and Campagna, D. P., "Comparison of Convergence Properties of CMAC Neural Network and Traditional Adaptive Controllers." Proc. 28th Conf. on Decision and Control, pp. 1744-1745, Tampa, Fla., December, 1989b.

[75] Kraft, L. G., An, E., and Campagna, D. P., "Comparison of CMAC Controller Weight Update Laws." Proc. 28th Conf. on Decision and Control, pp. 1746-1747, Tampa, Fla., December, 1989.

[76] Kraft, L. G., and Campagna, D. P., "A Summary Comparison of CMAC Neural Network and Traditional Adaptive Control Systems." IEEE Control Systems Magazine, April, 1990.

[77] Kraft, L. G., and Campagna, D. P., "Comparison of CMAC Architectures for Neural network Based control," Proceedings of the 29th IEEE Conference on Decision and control Part 6 (of 6), Honolulu, HI., Pp. 3267-3269, Dec. 5-7, 1990.

[78] Kraft, L. G., "Optimal Control Using CMAC Neural Networks." In *Neural Networks and Intelligent Control*, edited by D. A. White and D. A. Sofge, Van Nostrand-Reinhold, New York NY, Scheduled release in 1992.

[79] Kraft, L. G., Miller, W. T., and Dietz, D., "Development and Application of CMAC Neural Network-Based Control", In *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, edited by D. A. White and D. A. Sofge, Van Nostrand-Reinhold, New York NY, 1992.

[80] Kraft, L.G., An, E. and Ho, S., "Stability Properties of CMAC Neural Networks," Proceedings of the American Control Conference, pp. 1586-1591, 1991.

[81] Kuc, T.-Y. and Nam, K., "CMAC based iterative learning control of robot manipulators," 28th. IEEE Conference on Decision and Control, Tampa, FL, Vol. 3, pp. 2613-2618, 1989.

[82] Lane, S., Handelman, D., and Gelfand, J. J., "Higher-order CMAC neural networks-theory and practice," American Control Conference, Boston, MA, Vol. 2, pp. 1579-1585, 1991.

[83] Lane, S.H., Handelman, D. A., and Gelfand, J. J., "Theory and Development of

Higher-Order CMAC Neural Networks," IEEE Control Systems Magazine, Vol. 12, pp. 23-30, April 2, 1992

[84] Lee, J. and Kramer, B., "On-line fault monitoring and detection using an integrated learning and reasoning approach," Japan-USA Symposium on Flexible Automation, San Francisco, CA, Vol. 1, pp. 235-242, 1992.

[85] Lee, J. and Kramer, B. M., "Analysis of machine degradation using a neural network baded pattern discrimination model," Journal of Manufacturing Systems, Vol. 12, pp. 379-387, 1993

[86] Lin, Chun-Shin and Kim, Hyongsuk, "CMAC-Based Adaptive Critic Self-Learning Control," IEEE Transaction on Neural Networks, Vol. 2, No. 5, pp 530-533, September 1991.

[87] Lin, C.-S. and Kim, H., "Selection of learning parameters for CMAC-based adaptive critic learning," International Conference on Artificial Neural Networks in Engineering, pp. 153-160, 1992.

[88] Lin, Y. and Song, S.-M., "Kinematic control and coordination of walking machine motion using neural networks," 1991 IEEE International Joint Conference on Neural Networks - IJCNN '91, Singapore, Singapore, pp. 248-253, 1991.

[89] Linse, D.J. and Stengel, R. F., "Neural networks for Function Approximation in Nonlinear Control," Proceedings of the 1990 American control Conference, San Diego, CA., pp 674-679, May 23-25, 1990.

[90] Marr, D., "A Theory of Cerebellar Cortex," Journal of Physiology, v 202, p 437-479, 1969.

[91] Marshall, A. M., Ellison, D., Samson, W. B., and Swanston, M. T., "Adapting CMAC for Improved Adaptive Control," acquired via ftp from the University of Abertay Dundee.

[92] Marshall, A. M., Ellison, D., Samson, W. B., and Swanston, M. T., "A Technique for Adding Global Generalisation to CMAC," acquired via ftp from the University of Abertay Dundee.

[93] McCulloch, W. S. and Pitts, W. H., "A Logical Calculus of the IdeasImminent in Nervous Activity," Bull. Math. Biophy, 5:115-133, 1943.

[94] Militzer, J. and Parks, P. C, "Convergence Properties in Learning Control Systems." Automation and Remote Control, 50 (1989), No. 2 Part 2, 254-286.

[95] Miller, W. T., "Real-Time Neural network Control of a Biped Walking Robot," Control Systems Magazine, February, 1994, pp 41-48.

[96] Miller, W. T., Latham, P. J., and Scalera, S. M., "Bipedal Gait Adaptation for Walking with Dynamic Balance," Proceedings of the 1991 American Controls Conference, Boston, Mass., vol. 2, pp. 1603-1608, June, 1991

[97] Miller, W. T., and Aldrich, C. M., "Rapid Learning Using CMAC Neural Networks: Real Time Control of an Unstable System." Proceedings of the Fifth IEEE International Symposium on Intelligent Control, pp. 465-470, Phil., PA, Sept. 5-7, 1990.

[98] Miller, W. T., and Hewes, R. P. "Real time experiments in neural network based learning control during high speed, nonrepetitive robot operations". Proceedings of the Third IEEE International Symposium on Intelligent Control, Washington, D.C., August 24-26, 1988.

[99] Miller, W. T.: A Nonlinear Learning Controller for Robotic Manipulators. Proc of

the SPIE: Intelligent Robots and Computer Vision 726:416-423, 1986.

[100] Miller, W. T.: A Learning Controller for Nonrepetitive Robotic Operations. Proc. of the Workshop on Space Telerobotics, JPL Publication 87-13, II:273-281, Pasadena, CA, January 19-22, 1987.

[101] Miller, W. T., "Sensor based control of robotic manipulators using a general learning algorithm," IEEE Journal of Robotics and Automation RA-3:157-165, 1987b.

[102] Miller, W. T., Glanz, F. H., Kraft, L. G., "Application of a general learning algorithm to the control of robotic manipulators," International Journal of Robotics Research 6.2:84-98, 1987.

[103] Miller, W. T., "Real time learned sensor processing and motor control for a robot with vision," First Annual Conference of the International Neural Network Society, Boston, MA September 6-10, 1988, p. 347.

[104] Miller, W. T., "Real time application of neural networks for sensor-based control of robots with vision," IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Information Technology for Sensory-Based Robot Manipulators, vol. 19, pp. 825-831, July/August, 1989.

[105] Miller, W. T., Hewes, R. P., Glanz, F. H., and Kraft, L. G., "Real time dynamic control of an industrial manipulator using a neural network based learning controller," IEEE Journal of Robotics and Automation, vol. 6, pp. 1-9, 1990.

[106] Miller, W. T., An, E., Glanz, F. H., and Carter, M. J., "The Design of CMAC Neural Networks for Control." Proceedings of the Sixth Yale Workshop on Adaptive Systems, New Haven, CT, August 15-17, pp. 140-145, 1990.

[107] Miller, W. T., Glanz, F. H., and Kraft, L. G.: CMAC: An associative neural network alternative to backpropagation. Proceedings of the IEEE, Special Issue on Neural Networks, II, vol. 78, pp. 1561-1567, October, 1990.

[108] Miller, W. T., Sutton, R. S., and Werbos, P. J. (editors): *Neural Networks for Control*. Cambridge, MA, MIT Press, December, 1990.

[109] Miller, W. T., "The CMAC Architecture and Its Implementations." Presented at the workshop "From Theory to Prototype: Developing Relationships Between Universities and Industries in Eastern Europe and the United States," sponsored by the Digital Equipment Corporation, the University of New Hampshire, and the Technical University of Budapest, Budapest, Hungary, Sept. 4-6, 1991.

[110] Miller, W. T., Box, B. A., Whitney, E. C., and Glynn, J. M., "Design and implementation of a high speed CMAC neural network using programmable logic cell arrays." In Advances in Neural Information Processing Systems 3, edited by R.P. Lippmann, J.E. Moody, and D.S. Touretzky. Morgan Kaufmann, San Mateo, CA, pp. 1022-1027, 1991.

[111] Miller, W. T., Kraft, L. G., and Glanz, F. H., "Real time comparison of neural network and traditional adaptive controllers," The Yale Conference on Adaptive Control, May 20-22,1992, Yale University, New Haven, CT, pp. 99-104.

[112] Miller, W. T., "Real-time neural network control of a biped walking robot," IEEE Transactions on Automatic Control, 1993.

[113] Miller, W. T., "Real-time control of a biped walking robot," World Conference on Neural Networks, Portland, OR, pp. 1993.

[114] Miller, W. T., "Learning dynamic balance of a biped walking robot," IEEE International Conference on Neural Networks, Orlando, Florida, Vol. 5, pp. 2771-2776,

1994.

[115] Miller, W. T., Arehart, K. F., Scalera, S. M., and Gresham, H. L., "On-Line Hand-Printed Character Recognition Using CMAC Neural Networks," Internal Report 1994.

[116] Miller, W. T., Arehart, K. F., Scalera, S. M., and Gresham, H. L., "On-line hand-printed character recognition using CMAC neural networks," World Conference on Neural Networks, Portland, OR, July 12-15, 1993, pp. IV10-IV13.

[117] Minsky, Marvin L. and Papert, Seymour A., Perceptrons Expanded Edition, Cambridge, MA, MIT Press, 1969, 1988.

[118] Moody, J. and Darken, C., "Fast learning in networks of locally-tuned processing units," Neural Computation, Vol. 1, pp. 281-294, 1989.

[119] Moody, J. and Darken, C., "Learning with localized receptive fields," Connectionists Models Summer School, pp. 1988.

[120] Moody, J. and Darken, C., "Speedy alternatives to back propagation," International Neural Network Society First Annual Meeting, Boston, MA, pp. 202, 1988.

[121] Motor, T. "Machine Printed Character Recognition Using Multiple CMAC Neural Networks and Polar-Log Mapping for Feature Extraction," Masters Thesis, University of New Hampshire, 1995.

[122] Nelson, J., "Real-Time Control of a Robotic Pole Balancing System Using Complementary Neural Network and Optimal Techniques," Masters Thesis, University of New Hampshire, 1994.

[123] Nie, J.and Linkens, D. A., "Fuzzified CMAC self-learning controller," Second IEEE International Conference on Fuzzy Systems, San Francisco, CA, pp. 500-505, 1993.

[124] Norris, G., "Development and Control of a 6 Degree-of-Freedom Biped Walking Robot." M.S. Thesis, ECE Dept., U. of New Hampshire, Durham, NH, December, 1991.

[125] Ozawa, J., Hayashi, I., and Wakami, N., "Formulation of CMAC-fuzzy system," IEEE international Conference on Fuzzy Systems - Fuzz-IEEE, San Diego, CA, pp. 1179-1186, 1992.

[126] Park, H. and Cho, H. S., "CMAC-based learning controller for pressure tracking control of hydroforming processes," Winter Annual Meeting of the American Society of Mechanical Engineers, Dallas, TX, pp. 101-106, 1990.

[127] Parks, P. C. and Militzer, J., "Convergence Properties of Associative Memory Storage For Learning Control Systems," in IFAC Symposium on Adaptive Syst. in Control and Signal Processing, Glasgow, UK, 1989.

[128] Parks, P. C. and Militzer, J., "Convergence Properties of Associative Memory Storage For Learning Control Systems," in Automation and Remote Control, Plenum Press, New York. vol. 50, No. 2, 254-286, 1989.

[129] Parks, P. C. and Militzer, J., "Improved allocation of weights for associative memory storage in learning control systems," IFAC Design Methods of Control Systems, Zurich, Switzerland, pp. 507-512, 1991.

[130] Parks, P. C., Militzer, J. "A Comparison of Five Algorithms for the Training of CMAC Memories for Learning Control Systems," Automatica, Vol. 28, No. 5. pp 1027-1035, 1992.

[131] Peterson, J. K., "On-line estimation of optimal control sequences," International

Conference on Artificial Neural Networks in Engineering, pp. 579-584, 1992.

[132] Peterson, J. K. and Shelton, R. O., "Use of CMAC neural architectures in obstacle avoidance," 3rd. Workshop on Neural Networks: Academic/Industrial/NASA/ Defense, Alabama, AL, Vol. 1721, pp. 187-194, 1993.

[133] T. Poggio and F. Girosi, A Theory for Approximation and Learning, MIT AI Lab, Rept. no. No. 1140, July, 1989.

[134] Ramesh, N. and Sethi, I. K., "Nearest neighbor classification using CMAC," IEEE international Conference on Neural Networks, Orlando, Florida, Vol. 5, pp. 3061-3066, 1994.

[135] Reay, D. S., Green, T. C., and Williams, B. W., "Application of Associative memory Neural Networks to the Control of a Switched Reluctance Motor," Proc. IECON '93, Maui, HI, pp. 200-206, 1993.

[136] Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psych, Rev. 65: 386-408, 1958.

[137] Rosenblatt, F., *Principles of Neurodynamics*, New York, Spartan Books, 1962.

[138] Rudolph, F., "Locally Optimizing Neural Networks in Adaptive Robot Path Planning." Proc. IJCNN, Washington, D.C., Jan 15-19, 1990.

[139] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, vol 1, pp. 318-362, New York, American Institute of Physics, 1986.

[140] Sebald, A. V. and Schlenzig, J., "Minimax design of neural net controllers for highly uncertain plants," IEEE Transactions on Neural Networks, Vol. 5, pp. 73-82, 1994.

[141] Sebald, A. V., Sebald, C. A., and Schlenzig, J., "Use of neural net control strategies in difficult adaptive control problems closed loop control of drug infusion," 23rd. Annual Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, Vol. 1, pp. 342-345, 1989.

[142] Shelton, R.O. and Peterson, J.K., "Controlling a Truck with an Adaptive Critic CMAC Design," Simulation, Vol. 58, no. 5, pp 319-326, May 5, 1992.

[143] Shelton, R. O. and Peterson, J. K., "Controlling a truck with an adaptive critic temporal difference CMAC design," 3rd. Workshop on Neural Networks: Academic/ Industrial/NASA/Defense, Auburn, AL, SPIE Vol. 1721, pp. 195-206, 1993.

[144] Simpson, G. and Reinhard, K., A new approach to event location, UNH - GRO-Comptel Group, Rept. no. COM-TN-UNH-F70-044, June 9, 1988.

[145] Simpson, G. and Li, K., Artificial neural networks: solutions to problems in remote sensing, Earth Observation Sciences, Ltd (EOS), Rept. no. EOS-92/00(16000)-RP-001, March 1993

[146] Suwirjo, J., "A CMAC based hand-eye coordination system," Masters Thesis, University of New Hampshire, 1992.

[147] Tolle, H., Parks, P. C., and Ersu, E., "Learning Control with Interpolating Memories--General Ideas, Design layout, Theoretical Approaches and Practical Applications," International Journal of Control, vol 56, no 2, p 291, Aug 1992.

[148] Tolle, H. and Ersu, E., Neurocontrol. Berlin Heidelberg: Springer-Verlag, 1992.

[149] Verrall, D. and Simpson, G., Neural networks for meteosat cloud classification, Earth Observation Sciences, Ltd. (EOS), Rept. no. EOS-92/078-RP-001, Oct. 1992.

[150] Wasser, D. J.,Hislop, D. W., and Johnson, R. N., "Evaluation of a neural network

for fault-tolerant, real-time, adaptive control," Images of the Twenty-first Century - 11 Annual International Conference of the IEEE Engineering in Medicine and Biology, Seattle, WA, pp. 2027-2028, 1989.

[151] Wen, R.-C., et al., "A CMAC neural network chip for color correction," IEEE International Conference on Neural Networks, Orlando, Florida, Vol. 3, pp. 1943-1948, 1994.

[152] Werntges, H., "Delta rule-based neural networks for inverse kinematics," 1990 International Joint Conference on Neural Networks, San Diego, CA, Vol. 3, pp. 415-420, 1990.

[153] Wheeler, K., "Fault Tolerant Feedforward Control Using CMAC Neural Networks." M.S. Thesis, ECE Dept., Univ. of New Hampshire, Durham, NH, May, 1991.

[154] Widrow, B., and Hoff, M. E., "Adaptive Switching Circuits," IRE Western Electric Show and Convention Record, part 4, pp. 96-104, August, 23, 1960.

[155] Widrow, B. "Generalization and Information Storage in networks of Adaline 'Neurons'," In *Self-organizing Systems*. M. C. Jovitz, G. T. Jacobi, and G. Goldstein. eds., Washington, D. C., Spartan Books, 435-461, 1962.

[156] Wilson, E. D. and LaCourse, J.R., "Analyzing Biological Signals with CMAC, A Neural Network," Proceedings of the 17th Annual IEEE Northeast Bioengineering Conference, Hartford, CT, pp 3-4, April 4-5, 1991

[157] Wong, Y.-F., "CMAC Learning is Governed by a Single Parameter," IEEE Conf. on Neural Networks, San Francisco, CA, pp. 1439-1443, March, 1993.

[158] Wong, Y. and Sideris, A., "Learning Convergence in the Cerebellar Model Articulation Controller," IEEE Transactions on Neural Networks, Vol 3, No. 1, pp 115-121, January, 1992.

[159] Xu, L., Jiang, J.-P., and Zhu, J., "Supervised learning control of a nonlinear polymerization reactor using the CMAC neural network for knowledge storage," IEE Proceedings:Control Theory and Application, Vol. 141, pp. 33-38, 1994.

[160] Yao, S. and Bo, Z., "Learning convergence of CMAC in cyclic learning," International Joint Conference on Neural Networks, Nagoya, Jpn, Vol. 3, pp. 2583-2586, 1993

[161] Yang, B., "A VLSI Implementation of the CMAC Neural Network." MS Thesis, University of New Hampshire, 1992.

[162] Yang, X. J.,"Experimental parameter studies for the CMAC neural network," MS Thesis, University of New Hampshire, 1993.

[163] Zhu, J. J., Xiao, W., "Case Studies on CMAC Neural Network in the Control of Time-Varying Dynamical Systems," ICANNE, 1992.

References and Bibliography (Non-CMAC related)

[164] Axelby, G. S. and Parks, P. C., "Lyapunov Centenary," Automatica, vol 28, no 5, p 863, Sept. 1992

[165] Astrom, K. J., and Wittenmark, B., "On Self-tuning Regulators," Automatica, vol 9, pp 185-199, 1973.

[166] Astrom, K. J., Borrison, K. Ljung. L. and Wittenberg, B., "Theory and Applications of Self-tuning Regulators," Automatica, 1977.

[167] Astrom, K. J., and Wittenmark, B., *Adaptive Control*, Addison-Wesley, 1995.

[168] Carroll, R. L. and Lindorff, D. P., "An Adaptive Observer for Single-Input Single-Output Linear systems," IEEE Transactions on Automatic Control, AC-18, No. 5, October, 1973.

[169] Chen, C. T., *Introduction to Linear System Theory*, Hold, Rinehart and Winston, Inc. New York, New York.

[170] Dudgeon, D. E. and Mersereau, R. M., Multidimensional Digital Signal Processing. Prentice-Hall, Inc., 1984.

[171] Eykhoff, P. and Parks, P. C., "Identification and System Parameter Estimation: Where Do We Stand Now?" Automatica, vol 26, no 1, p 3, Jan 1990.

[172] Joel Franklin, *Matrix Theory*, Prentice-Hall, 1968

[173] M. I. Friedlin and A. D. Wentzell, *Random Perturbations of Dynamical Systems*, New York: Springer-Verlag, 1984.

[174] G. C. Goodwin and D. Q. Mayne, "A parameter estimation perspective of continuous-time model reference adaptive control," Automatica, vol. 23, no 1, pp. 57-70, 1987.

[175] Kalman, R. E., "Design of a Self Optimizing Control System," Transactions of the ASME, 80, pp. 468-478, 1958.

[176] Kirk, D. E., *Optimal Control Theory: An Introduction*, Prentice-Hall, Englewood Cliffs New Jersey, 1970.

[177] Kreisselmeier, G. "Adaptive Observers with Exponential Rate of convergence," IEEE Transactions on Automatic Control, Vol. AC-22, No. 1. February 1977, pp. 2-8.

[178] Kudva, P. and Narendra, K. S., "Synthesis of an Adaptive Observer Using Liapunov's Direct Method," Becton Center Technical Report, CT-55, Yale University, New Haven, CT, March, 1973.

[179] Kushner, H. J., "On the Convergence of Lion's Identification Method with Random inputs," IEEE Transactions on Automatic Control, Vol. AC-15, No. 6, December 1970, pp. 652-654.

[180] Monopoli, R., "Liapunov's Method for Adaptive Control-System Design," correspondence re: Parks [186], in IEEE Trans. on Automatic Control, pp. 334-335, June, 1967.

[181] Monopoli, R., "Model Reference Adaptive Control with Augmented Error Signal," IEEE Transactions on Automatic control, vol. AC-19, No. 5, October 1974, pp. 474-484.

[182] Moody, J. and C. Darken, "Learning with Localized Receptive Fields," Yale University Research Report YALEU/DCS/RR- 649, September, 1988.

[183] Narendra, K. S. and Annaswamy, A. M., *Stable Adaptive Systems*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[184] Narendra, K. S. and Parthasarathy, K, "Identification and Control of Dynamical Systems Using Neural Networks," IEEE Trans. on Neural Networks, Vol. 1, No. 1, pp 4-27, March, 1990.

[185] Oppenheim, A. V. and Schaferr, R. W. , Discrete-Time Signal Processing. Englewood Cliffs, N.J.: Prentice Hall, 1989.

[186] Parks, P. C., "Liapunov Redesign of Model Reference Adaptive Control System," IEEE Transactions on Automatic control, vol. AC-11, No. 3, July 1966

[187] Parthasarathy, K. and Narendra, K. S., "Stable adaptive control of a class of dis-

crete-time nonlinear systems using radial basis neural networks." Technical report, Center for systems Science, Yale University, New Haven, CT, February 1991.

[188] Pereiras, A., Kim, C. H. and Lindorff, D. P., "Convergence Properties of Adaptive Observers," IEEE Conference on Decision and Control, 1974, pp. 295-300.

[189] Peterson, D. and Middleton, D., "Sampling and reconstruction of wave-number limited functions in N-dimensional Euclidean spaces," Information and Control, Vol. 5, pp. 279-323, 1962.

[190] Poggio, T., and Girosi, F., "Networks for Approximation and Learning." Proc. IEEE, Vol. 78 (9), pp. 1481- 1497, September 1990.

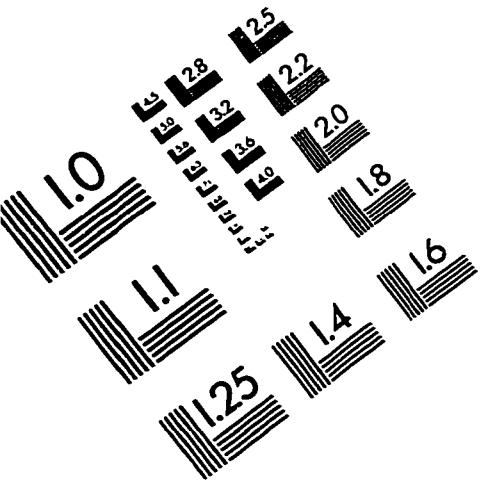
[191] Sanders, J. A. and Verhulst, F., *Averaging Methods in Nonlinear Dynamical Systems*, New York: Springer-Verlag, 1985.

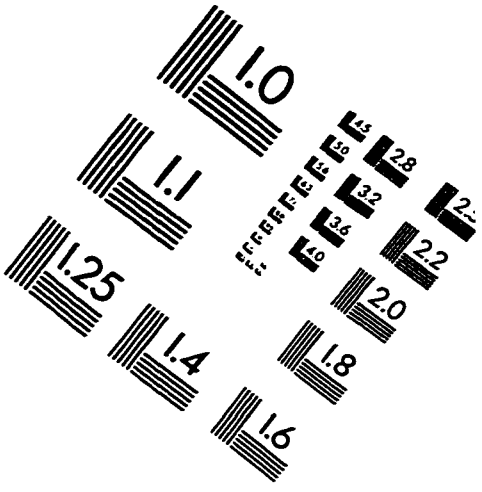
[192] Sontag, E. D., *Mathematical Control Theory*, Springer-Verlag, New York, 1989

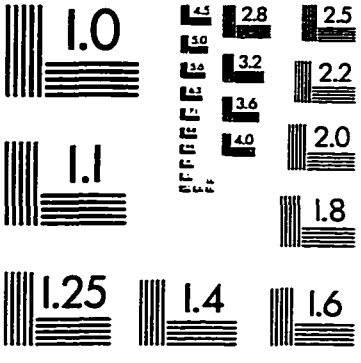
[193] Werbos, P. J., "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", Doctoral Dissertation, Harvard University, August 1974.

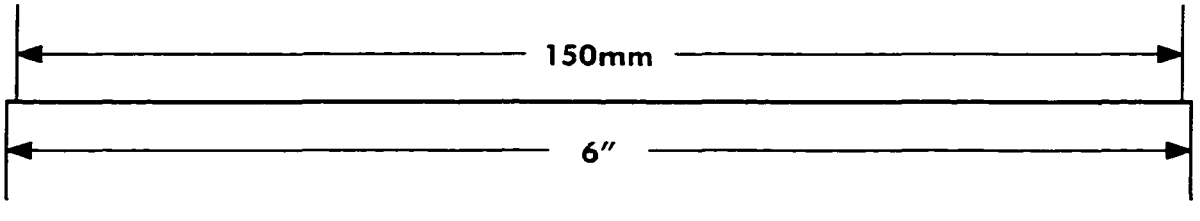
[194] Osburn, P. V., Whitaker, H. P., and Kezer, A, "New Developments in the Design of Model Reference Adaptive Control Systems,", in Proceedings of the IAS 29th Annual Meeting, New York, NY, 1961.

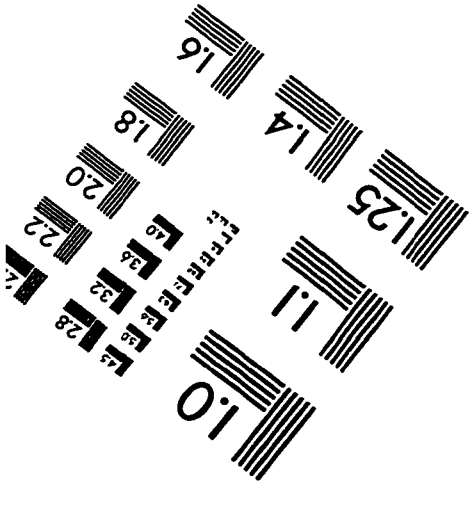
IMAGE EVALUATION TEST TARGET (QA-3)











APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989