Computational Intelligence in Control

Masoud Mohammadian, University of Canberra, Australia
Ruhul Amin Sarker, University of New South Wales, Australia
Xin Yao, University of Birmingham, UK

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore • Beijing

Acquisition Editor:	Mehdi Khosrowpour
Senior Managing Editor:	Jan Travers
Managing Editor:	Amanda Appicello
Development Editor:	Michele Rossi
Copy Editor:	Maria Boyer
Typesetter:	Tamara Gillis
Cover Design:	Integrated Book Technology
Printed at:	Integrated Book Technology

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Mohammadian, Masoud.
Computational intelligence in control / Masoud Mohammadian, Ruhul Amin Sarker and Xin Yao.
p. cm.
ISBN 1-59140-037-6 (hardcover) -- ISBN 1-59140-079-1 (ebook)
1. Neural networks (Computer science) 2. Automatic control. 3. Computational intelligence. I. Amin, Ruhul. II. Yao, Xin, 1962- III. Title.
QA76.87 .M58 2003
006.3--dc21

2002014188

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Computational Intelligence in Control

Table of Contents

Preface

This book covers the recent applications of computational intelligence techniques for modelling, control and automation. The application of these techniques has been found useful in problems when the process is either difficult to model or difficult to solve by conventional methods. There are numerous practical applications of computational intelligence techniques in modelling, control, automation, prediction, image processing and data mining.

Research and development work in the area of computational intelligence is growing rapidly due to the many successful applications of these new techniques in very diverse problems. "Computational Intelligence" covers many fields such as neural networks, (adaptive) fuzzy logic, evolutionary computing, and their hybrids and derivatives. Many industries have benefited from adopting this technology. The increased number of patents and diverse range of products developed using computational intelligence methods is evidence of this fact.

These techniques have attracted increasing attention in recent years for solving many complex problems. They are inspired by nature, biology, statistical techniques, physics and neuroscience. They have been successfully applied in solving many complex problems where traditional problem-solving methods have failed. These modern techniques are taking firm steps as robust problem-solving mechanisms.

This volume aims to be a repository for the current and cutting-edge applications of computational intelligent techniques in modelling control and automation, an area with great demand in the market nowadays.

With roots in modelling, automation, identification and control, computational intelligence techniques provide an interdisciplinary area that is concerned with learning and adaptation of solutions for complex problems. This instantiated an enormous amount of research, searching for learning methods that are capable of controlling novel and non-trivial systems in different industries.

This book consists of open-solicited and invited papers written by leading researchers in the field of computational intelligence. All full papers have been peer review by at least two recognised reviewers. Our goal is to provide a book

that covers the foundation as well as the practical side of the computational intelligence.

The book consists of 17 chapters in the fields of self-learning and adaptive control, robotics and manufacturing, machine learning, evolutionary optimisation, information retrieval, fuzzy logic, Bayesian systems, neural networks and hybrid evolutionary computing.

This book will be highly useful to postgraduate students, researchers, doctoral students, instructors, and partitioners of computational intelligence techniques, industrial engineers, computer scientists and mathematicians with interest in modelling and control.

We would like to thank the senior and assistant editors of Idea Group Publishing for their professional and technical assistance during the preparation of this book. We are grateful to the unknown reviewers for the book proposal for their review and approval of the book proposal. Our special thanks goes to Michele Rossi and Mehdi Khosrowpour for their assistance and their valuable advise in finalizing this book.

We would like to acknowledge the assistance of all involved in the collation and review process of the book, without whose support and encouragement this book could not have been successfully completed.

We wish to thank all the authors for their insights and excellent contributions to this book. We would like also to thank our families for their understanding and support throughout this book project.

M. Mohammadian, R. Sarker and X. Yao

SECTION I:

NEURAL NETWORKS DESIGN, CONTROL AND ROBOTICS APPLICATION

Chapter I

Designing Neural Network Ensembles by Minimising Mutual Information

Yong Liu
The University of Aizu, Japan

Xin Yao
The University of Birmingham, UK

Tetsuya Higuchi
National Institute of Advanced Industrial Science and Technology, Japan

ABSTRACT

This chapter describes negative correlation learning for designing neural network ensembles. Negative correlation learning has been firstly analysed in terms of minimising mutual information on a regression task. By minimising the mutual information between variables extracted by two neural networks, they are forced to convey different information about some features of their input. Based on the decision boundaries and correct response sets, negative correlation learning has been further studied on two pattern classification problems. The purpose of examining the decision boundaries and the correct response sets is not only to illustrate the learning behavior of negative correlation learning, but also to cast light on how to design more effective neural network ensembles. The experimental results showed the decision boundary of the trained neural network ensemble by negative correlation learning is almost as good as the optimum decision boundary.

INTRODUCTION

In single neural network methods, the neural network learning problem is often formulated as an optimisation problem, i.e., minimising certain criteria, e.g., minimum error, fastest learning, lowest complexity, etc., about architectures. Learning algorithms, such as backpropagation (BP) (Rumelhart, Hinton & Williams, 1986), are used as optimisation algorithms to minimise an error function. Despite the different error functions used, these learning algorithms reduce a learning problem to the same kind of optimisation problem.

Learning is different from optimisation because we want the learned system to have best generalisation, which is different from minimising an error function. The neural network with the minimum error on the training set does not necessarily have the best generalisation unless there is an equivalence between generalisation and the error function. Unfortunately, measuring generalisation exactly and accurately is almost impossible in practice (Wolpert, 1990), although there are many theories and criteria on generalisation, such as the minimum description length (Rissanen, 1978), Akaike's information criteria (Akaike, 1974) and minimum message length (Wallace & Patrick, 1991). In practice, these criteria are often used to define better error functions in the hope that minimising the functions will maximise generalisation. While better error functions often lead to better generalisation of learned systems, there is no guarantee. Regardless of the error functions used, single network methods are still used as optimisation algorithms. They just optimise different error functions. The nature of the problem is unchanged.

While there is little we can do in single neural network methods, there are opportunities in neural network ensemble methods. Neural network ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, a neural network ensemble combines a set of neural networks which learn to subdivide the task and thereby solve it more efficiently and elegantly. A neural network ensemble offers several advantages over a monolithic neural network. First, it can perform more complex tasks than any of its components (i.e., individual neural networks in the ensemble). Secondly, it can make an overall system easier to understand and modify. Finally, it is more robust than a monolithic neural network and can show graceful performance degradation in situations where only a subset of neural networks in the ensemble are performing correctly. Given the advantages of neural network ensembles and the complexity of the problems that are beginning to be investigated, it is clear that the neural network ensemble method will be an important and pervasive problem-solving technique.

The idea of designing an ensemble learning system consisting of many subsystems can be traced back to as early as 1958 (Selfridge, 1958; Nilsson, 1965). Since the early 1990s, algorithms based on similar ideas have been developed in many different but related forms, such as neural network ensembles

(Hansen & Salamon, 1990; Sharkey, 1996), mixtures of experts (Jacobs, Jordan, Nowlan & Hinton, 1991; Jacobs & Jordan, 1991; Jacobs, Jordan & Barto, 1991; Jacobs, 1997), various boosting and bagging methods (Drucker, Cortes, Jackel, LeCun & Vapnik, 1994; Schapire, 1990; Drucker, Schapire & Simard, 1993) and many others. There are a number of methods of designing neural network ensembles. To summarise, there are three ways of designing neural network ensembles in these methods: independent training, sequential training and simultaneous training.

A number of methods have been proposed to train a set of neural networks independently by varying initial random weights, the architectures, the learning algorithm used and the data (Hansen et al., 1990; Sarkar, 1996). Experimental results have shown that networks obtained from a given network architecture for different initial random weights often correctly recognize different subsets of a given test set (Hansen et al., 1990; Sarkar, 1996). As argued in Hansen et al. (1990), because each network makes generalisation errors on different subsets of the input space, the collective decision produced by the ensemble is less likely to be in error than the decision made by any of the individual networks.

Most independent training methods emphasised independence among individual neural networks in an ensemble. One of the disadvantages of such a method is the loss of interaction among the individual networks during learning. There is no consideration of whether what one individual learns has already been learned by other individuals. The errors of independently trained neural networks may still be positively correlated. It has been found that the combining results are weakened if the errors of individual networks are positively correlated (Clemen & Winkler, 1985). In order to decorrelate the individual neural networks, sequential training methods train a set of networks in a particular order (Drucker et al., 1993; Opitz & Shavlik, 1996; Rosen, 1996). Drucker et al. (1993) suggested training the neural networks using the boosting algorithm. The boosting algorithm was originally proposed by Schapire (1990). Schapire proved that it is theoretically possible to convert a weak learning algorithm that performs only slightly better than random guessing into one that achieves arbitrary accuracy. The proof presented by Schapire (1990) is constructive. The construction uses filtering to modify the distribution of examples in such a way as to force the weak learning algorithm to focus on the harder-to-learn parts of the distribution.

Most of the independent training methods and sequential training methods follow a two-stage design process: first generating individual networks, and then combining them. The possible interactions among the individual networks cannot be exploited until the integration stage. There is no feedback from the integration stage to the individual network design stage. It is possible that some of the independently designed networks do not make much contribution to the integrated system. In

order to use the feedback from the integration, simultaneous training methods train a set of networks together. Negative correlation learning (Liu & Yao, 1998a, 1998b, 1999) and the mixtures-of-experts (ME) architectures (Jacobs et al., 1991; Jordan & Jacobs, 1994) are two examples of simultaneous training methods. The idea of negative correlation learning is to encourage different individual networks in the ensemble to learn different parts or aspects of the training data, so that the ensemble can better learn the entire training data. In negative correlation learning, the individual networks are trained simultaneously rather than independently or sequentially. This provides an opportunity for the individual networks to interact with each other and to specialise.

In this chapter, negative correlation learning has been firstly analysed in terms of minimising mutual information on a regression task. The similarity measurement between two neural networks in an ensemble can be defined by the explicit mutual information of output variables extracted by two neural networks. The mutual information between two variables, output F_i of network i and output F_j of network j, is given by

$$I(F_i; F_j) = h(F_i) + h(F_j) - h(F_i, F_j) \tag{1}$$

where $h(F_i)$ is the entropy of F_i, $h(F_j)$ is the entropy of F_j, and $h(F_i, F_j)$ is the joint differential entropy of F_i and F_j. The equation shows that joint differential entropy can only have high entropy if the mutual information between two variables is low, while each variable has high individual entropy. That is, the lower mutual information two variables have, the more different they are. By minimising the mutual information between variables extracted by two neural networks, they are forced to convey different information about some features of their input. The idea of minimising mutual information is to encourage different individual networks to learn different parts or aspects of the training data so that the ensemble can learn the whole training data better.

Based on the decision boundaries and correct response sets, negative correlation learning has been further studied on two pattern classification problems. The purpose of examining the decision boundaries and the correct response sets is not only to illustrates the learning behavior of negative correlation learning, but also to cast light on how to design more effective neural network ensembles. The experimental results showed the decision boundary of the trained neural network ensemble by negative correlation learning is almost as good as the optimum decision boundary.

The rest of this chapter is organised as follows: Next, the chapter explores the connections between the mutual information and the correlation coefficient, and

explains how negative correlation learning can be used to minimise mutual information; then the chapter analyses negative correlation learning via the metrics of mutual information on a regression task; the chapter then discusses the decision boundaries constructed by negative correlation learning on a pattern classification problem; finally the chapter examines the correct response sets of individual networks trained by negative correlation learning and their intersections, and the chapter concludes with a summary of the chapter and a few remarks.

MINIMISING MUTUAL INFORMATION BY NEGATIVE CORRELATION LEARNING

Minimisation of Mutual Information

Suppose the output F_i of network i and the output F_j of network j are Gaussian random variables. Their variances are σ_i^2 and σ_j^2, respectively. The mutual information between F_i and F_j can be defined by Eq.(1) (van der Lubbe, 1997, 1999). The differential entropy $h(F_i)$ and $h(F_j)$ are given by

$$h(F_i) = [1 + log(2\pi\sigma_i^2)] / 2 \tag{2}$$

and

$$h(F_j) = [1 + log(2\pi\sigma_j^2)] / 2 \tag{3}$$

The joint differential entropy $h(F_i, F_j)$ is given by

$$h(F_i, F_j) = 1 + log(2\pi) + log|det(\Sigma)| \tag{4}$$

where Σ is the 2-by-2 covariance matrix of F_i and F_j. The determinant of Σ is

$$det(\Sigma) = \sigma_i^2\sigma_j^2(1 - \rho_{ij}^2) \tag{5}$$

where ρ_{ij} is the correlation coefficient of F_i and F_j

$$\rho_{ij} = E[(F_i - E[F_i])(F_j - E[F_j])] / (\sigma_i^2 \sigma_j^2) \quad (6)$$

Using the formula of Eq.(5), we get

$$h(F_i, F_j) = 1 + log(2\pi) + log[\sigma_i^2 \sigma_j^2 (1 - \rho_{ij}^2)] / 2 \quad (7)$$

By substituting Eqs.(2), (3), and (7) in (1), we get

$$I(F_i ; F_j) = - log(1 - \rho_{ij}^2) / 2 \quad (8)$$

From Eq.(8), we may make the following statements:

1. If F_i and F_j are uncorrelated, the correlation coefficient ρ_{ij} is reduced to zero, and the mutual information $I(F_i ; F_j)$ becomes very small.
2. If F_i and F_j are highly positively correlated, the correlation coefficient ρ_{ij} is close to 1, and mutual information $I(F_i ; F_j)$ becomes very large.

Both theoretical and experimental results (Clemen et al., 1985) have indicated that when individual networks in an ensemble are unbiased, average procedures are most effective in combining them when errors in the individual networks are negatively correlated and moderately effective when the errors are uncorrelated. There is little to be gained from average procedures when the errors are positively correlated. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of the population should be minimised. Minimising the mutual information between each individual neural network and the rest of the population is equivalent to minimising the correlation coefficient between them.

Negative Correlation Learning

Given the training data set $D = \{(\mathbf{x}(1),\mathbf{y}(1)), \ldots, (\mathbf{x}(N),\mathbf{y}(N))\}$, we consider estimating $\mathbf{y}$ by forming a neural network ensemble whose output is a simple averaging of outputs F_i of a set of neural networks. All the individual networks in the ensemble are trained on the same training data set D

$$F(n) = \frac{1}{M} \sum_{i=1}^{M} F_i(n) \quad (9)$$

where $F_i(n)$ is the output of individual network i on the nth training pattern $\mathbf{x}(n)$, $F(n)$ is the output of the neural network ensemble on the nth training pattern, and M is the number of individual networks in the neural network ensemble.

The idea of negative correlation learning is to introduce a correlation penalty term into the error function of each individual network so that the individual network can be trained simultaneously and interactively. The error function E_i for individual i on the training data set D in negative correlation learning is defined by

$$E_i = \frac{1}{N} \sum_{n=1}^{N} \left[\frac{1}{2} (F_i(n) - y(n))^2 + \lambda p_i(n) \right] \tag{10}$$

where N is the number of training patterns, $E_i(n)$ is the value of the error function of network i at presentation of the nth training pattern and $y(n)$ is the desired output of the nth training pattern. The first term in the right side of Eq.(10) is the mean-squared error of individual network i. The second term p_i is a correlation penalty function. The purpose of minimising p_i is to negatively correlate each individual's error with errors for the rest of the ensemble. The parameter λ is used to adjust the strength of the penalty.

The penalty function p_i has the form

$$p_i(n) = -(F_i(n)-F(n))^2 / 2 \tag{11}$$

The partial derivative of E_i with respect to the output of individual i on the nth training pattern is

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F_i(n) - y(n) - \lambda(F_i(n) - F(n)) \tag{12}$$

where we have made use of the assumption that the output of ensemble $F(n)$ has constant value with respect to $F_i(n)$. The value of parameter λ lies inside the range $0 \le \lambda \le 1$ so that both $(1-\lambda)$ and λ have nonnegative values. BP (Rumelhart et al., 1996) algorithm has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eq.(12) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an *epoch*. Negative correlation learning from Eq.(12) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda(F_i(n) - F(n))$ for the ith neural network.

From Eqs.(10), (11) and (12), we may make the following observations:

1. During the training process, all the individual networks interact with each other through their penalty terms in the error functions. Each network F_i minimises not only the difference between $F_i(n)$ and $y(n)$, but also the difference between $F(n)$ and $y(n)$. That is, negative correlation learning considers errors what all other neural networks have learned while training a neural network.
2. For $\lambda=0.0$, there are no correlation penalty terms in the error functions of the individual networks, and the individual networks are just trained independently using BP. That is, independent training using BP for the individual networks is a special case of negative correlation learning.
3. For $\lambda=1$, from Eq.(12) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - y(n) \tag{13}$$

Note that the error of the ensemble for the nth training pattern is defined by

$$E_{ensemble} = \frac{1}{2}\left(\frac{1}{M}\sum_{i=1}^{M} F_i(n) - y(n)\right)^2 \tag{14}$$

The partial derivative of $E_{ensemble}$ with respect to F_i on the nth training pattern is

$$\frac{\partial E_{ensemble}}{\partial F_i(n)} = \frac{1}{M}(F(n) - y(n)) \tag{15}$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ensemble}}{\partial F_i(n)} \tag{16}$$

The minimisation of the error function of the ensemble is achieved by minimising the error functions of the individual networks. From this point of view, negative correlation learning provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual networks.

ANALYSIS BASED ON MEASURING MUTUAL INFORMATION

In order to understand why and how negative correlation learning works, this section analyses it through measuring mutual information on a regression task in three cases: noise-free condition, small noise condition and large noise condition.

Simulation Setup

The regression function investigated here is

$$f(x) = \frac{1}{13}[10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5)] - 1 \quad (17)$$

where $\mathbf{x}=[x_1, ..., x_5]$ is an input vector whose components lie between zero and one. The value of $f(\mathbf{x})$ lies in the interval [-1, 1]. This regression task has been used by Jacobs (1997) to estimate the bias of mixture-of-experts architectures and the variance and covariance of experts' weighted outputs.

Twenty-five training sets, $(x^{(k)}(l), y^{(k)}(l))$, $l=1, ..., L$, $L=500$, $k=1, ..., K$, $K=25$, were created at random. Each set consisted of 500 input-output patterns in which the components of the input vectors were independently sampled from a uniform distribution over the interval (0, 1). In the noise-free condition, the target outputs were not corrupted by noise; in the small noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2=0.1$ to the function $f(\mathbf{x})$; in the large noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2=0.2$ to the function $f(\mathbf{x})$. A testing set of 1,024 input-output patterns, $(\mathbf{t}(n), d(n))$, $n = 1, ..., N$, $N = 1024$, was also generated. For this set, the components of the input vectors were independently sampled from a uniform distribution over the interval (0, 1), and the target outputs were not corrupted by noise in all three conditions. Each individual network in the ensemble is a multi-layer perceptron with one hidden layer. All the individual networks have 5 hidden nodes in an ensemble architecture. The hidden node function is defined by the logistic function

$$\varphi(y) = \frac{1}{1+\exp(-y)} \quad (18)$$

The network output is a linear combination of the outputs of the hidden nodes.

For each estimation of mutual information among an ensemble, 25 simulations were conducted. In each simulation, the ensemble was trained on a different training set from the same initial weights distributed inside a small range so that different simulations of an ensemble yielded different performances solely due to the use of different training sets. Such simulation setup follows the suggestions from Jacobs (1997).

Measurement of Mutual Information

The average outputs of the ensemble and the individual network i on the nth pattern in the testing set, (t(n), d(n)), n = 1, ..., N, are denoted and given respectively by

$$\overline{F}(\mathrm{t}(n)) = \frac{1}{K}\sum_{k=1}^{K} F^{(k)}(\mathrm{t}(n)) \tag{19}$$

and

$$\overline{F}_i(\mathrm{t}(n)) = \frac{1}{K}\sum_{k=1}^{K} F_i^{(k)}(\mathrm{t}(n)) \tag{20}$$

where $F^{(k)}(t(n))$ and $F_i^{(k)}(t(n))$ are the outputs of the ensemble and the individual network i on the nth pattern in the testing set from the kth simulation, respectively, and $K=25$ is the number of simulations. From Eq.(6), the correlation coefficient between network i and network j is given by

$$\rho_{ij} = \frac{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_i^{(k)}(\mathrm{t}(n)) - \overline{F}_i(\mathrm{t}(n))\right)\left(F_j^{(k)}(\mathrm{t}(n)) - \overline{F}_j(\mathrm{t}(n))\right)}{\sqrt{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_i^{(k)}(\mathrm{t}(n)) - \overline{F}_i(\mathrm{t}(n))\right)^2 \sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_j^{(k)}(\mathrm{t}(n)) - \overline{F}_j(\mathrm{t}(n))\right)^2}} \tag{21}$$

From Eq.(8), the integrated mutual information among the ensembles can be defined by

$$E_{mi} = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1, j\neq i}^{M} log(1-\rho_{ij}^2) \tag{22}$$

We may also define the integrated mean-squared error (MSE) on the testing set as

$$E_{mse} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{K}\sum_{k=1}^{K}(F^{(k)}(\mathrm{t}(n)) - d(n))^2 \quad (23)$$

The integrated mean-squared error E_{train_mse} on the training set is given by

$$E_{train\_mis} = \frac{1}{L}\sum_{l=1}^{L}\frac{1}{K}\sum_{k=1}^{K}(F^{(k)}(\mathrm{x}^{(k)}(l)) - y^{(k)}(l))^2 \quad (24)$$

Results in the Noise-Free Condition

The results of negative correlation learning in the noise-free condition for the different values of λ at epoch 2000 are given in Table 1. The results suggest that both E_{train_mse} and E_{test_mse} appeared to decrease with the increasing value of λ. The mutual information E_{mi} among the ensemble decreased as the value of λ increased when $0 \le \lambda \le 0.5$. However, when λ increased further to 0.75 and 1, the mutual information E_{mi} had larger values. The reason of having larger mutual information at $\lambda = 0.75$ and $\lambda = 1$ is that some correlation coefficients had negative values and the mutual information depends on the absolute values of correlation coefficients.

In order to find out why E_{train_mse} decreased with increasing value of λ, the concept of capability of a trained ensemble is introduced. The capability of a trained ensemble is measured by its ability of producing correct input-output mapping on the training set used, specifically, by its integrated mean-squared error E_{train_mse} on the training set. The smaller E_{train_mse} is, the larger capability the trained ensemble has.

Results in the Noise Conditions

Table 2 and Table 3 compare the performance of negative correlation learning for different strength parameters in both small noise (variance $\sigma^2 = 0.1$) and large

Table 1: The results of negative correlation learning in the noise-free condition for different l values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	0.3706	0.1478	0.1038	0.1704	0.6308
$E_{test\ mse}$	0.0016	0.0013	0.0011	0.0007	0.0002
$E_{train\ mse}$	0.0013	0.0010	0.0008	0.0005	0.0001

noise (variance $\sigma^2 = 0.2$) conditions. The results show that there were same trends for E_{mi}, E_{test_mse} and E_{train_mse} in both noise-free and noise conditions when $\lambda \leq 0.5$. That is, E_{mi}, E_{test_mse} and E_{train_mse} appeared to decrease with the increasing value of λ. However, E_{test_mse} appeared to decrease first and then increase with the increasing value of λ.

In order to find out why E_{test_mse} showed different trends in noise-free and noise conditions when $\lambda = 0.75$ and $\lambda = 1$, the integrated mean-squared error E_{train_mse} on the training set was also shown in Tables 1, 2 and 3. When $\lambda = 0$, the neural network ensemble trained had relatively large E_{train_mse}. It indicated that the capability of the neural network ensemble trained was not big enough to produce correct input-output mapping (i.e., it was underfitting) for this regression task. When $\lambda = 1$, the neural network ensemble trained learned too many specific input-output relations (i.e., it was overfitting), and it might memorise the training data and therefore be less able to generalise between similar input-output patterns. Although the overfitting was not observed for the neural network ensemble used in the noise-free condition, too large capability of the neural network ensemble will lead to overfitting for both noise-free and noise conditions because of the ill-posedness of any finite training set (Friedman, 1994).

Choosing a proper value of λ is important, and also problem dependent. For the noise conditions used for this regression task and the ensemble architectured used, the performance of the ensemble was optimal for $\lambda = 0.5$ among the tested values of λ in the sense of minimising the MSE on the testing set.

Table 2: The results of negative correlation learning in the small noise condition for different λ values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	6.5495	3.8761	1.4547	0.3877	0.2431
E_{test_mse}	0.0137	0.0128	0.0124	0.0126	0.0290
E_{train_mse}	0.0962	0.0940	0.0915	0.0873	0.0778

Table 3: The results of negative correlation learning in the large noise condition for different λ values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	6.7503	3.9652	1.6957	0.4341	0.2030
E_{test_mse}	0.0249	0.0235	0.0228	0.0248	0.0633
E_{train_mse}	0.1895	0.1863	0.1813	0.1721	0.1512

ANALYSIS BASED ON DECISION BOUNDARIES

This section analyses the decision boundaries constructed by both negative correlation learning and the independent training. The independent training is a special case of negative correlation learning for $\lambda = 0.0$ in Eq.(12).

Simulation Setup

The objective of the pattern classification problem is to distinguish between two classes of overlapping, two-dimensional, Gaussian-distributed patterns labeled 1 and 2. Let Class 1 and Class 2 denote the set of events for which a random vector **x** belongs to patters 1 and 2, respectively. We may then express the conditional probability density functions for the two classes:

$$f_X(x) = \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{1}{2\sigma_1^2}||x - \mu_1||^2\right) \tag{25}$$

where mean vector $\mu_1 = [0,0]^T$ and variance $\sigma_1^2 = 1$.

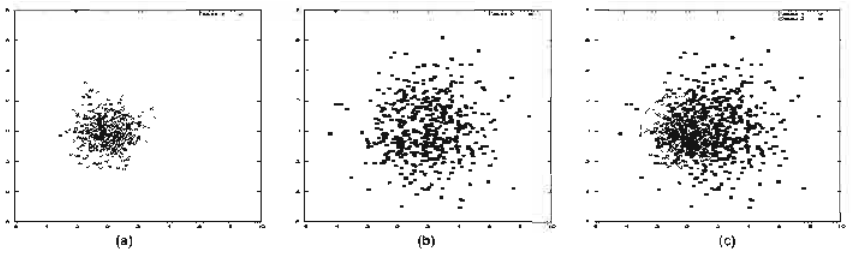
$$f_X(x) = \frac{1}{2\pi\sigma_2^2} \exp\left(-\frac{1}{2\sigma_2^2}||x - \mu_2||^2\right) \tag{26}$$

where mean vector $\mu_2 = [0,0]^T$ and variance $\sigma_2^2 = 4$. The two classes are assumed to be equiprobable; that is $p_1 = p_2 = ½$. The costs for misclassifications are assumed to be equal, and the costs for correct classifications are assumed to be zero. On this basis, the (optimum) Bayes classifier achieves a probability of correct classification $p_c = 81.51$ percent. The boundary of the Bayes classifier consists of a circle of center $[-2/3, 0]^T$ and radius $r = 2.34$; 1000 points from each of two processes were generated for the training set. The testing set consists of 16,000 points from each of two classes.

Figure 1 shows individual scatter diagrams for classes and the joint scatter diagram representing the superposition of scatter plots of 500 points from each of two processes. This latter diagram clearly shows that the two distributions overlap each other significantly, indicating that there is inevitably a significant probability of misclassification.

The ensemble architecture used in the experiments has three networks. Each individual network in the ensemble is a multi-layer perceptron with one hidden layer. All the individual networks have three hidden nodes in an ensemble architecture.

Figure 1: (a) Scatter plot of Class 1; (b) Scatter plot of Class 2; (c) Combined scatter plot of both classes; the circle represents the optimum Bayes solution



Both hidden node function and output node function are defined by the logistic function in Eq.(18).

Experimental Results

The results presented in Table 4 pertain to 10 different runs of the experiment, with each run involving the use of 2,000 data points for training and 32,000 for testing. Figures 2 and 3 compare the decision boundaries constructed by negative

Figure 2: Decision boundaries formed by the different networks trained by the negative correlation learning ($\lambda = 0.75$): (a) Network 1; (b) Network 2; (c) Network 3; (d) Ensemble; the circle represents the optimum Bayes solution

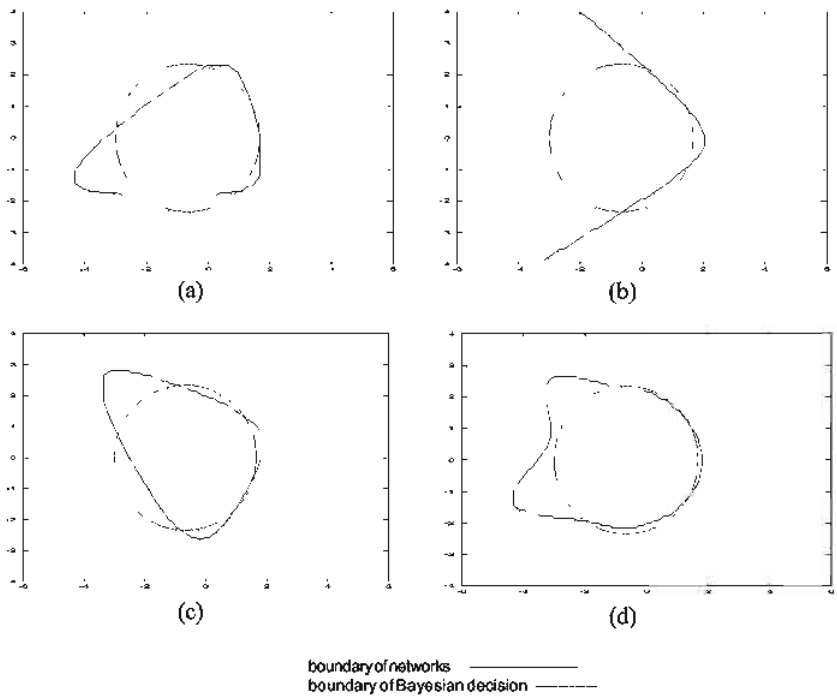
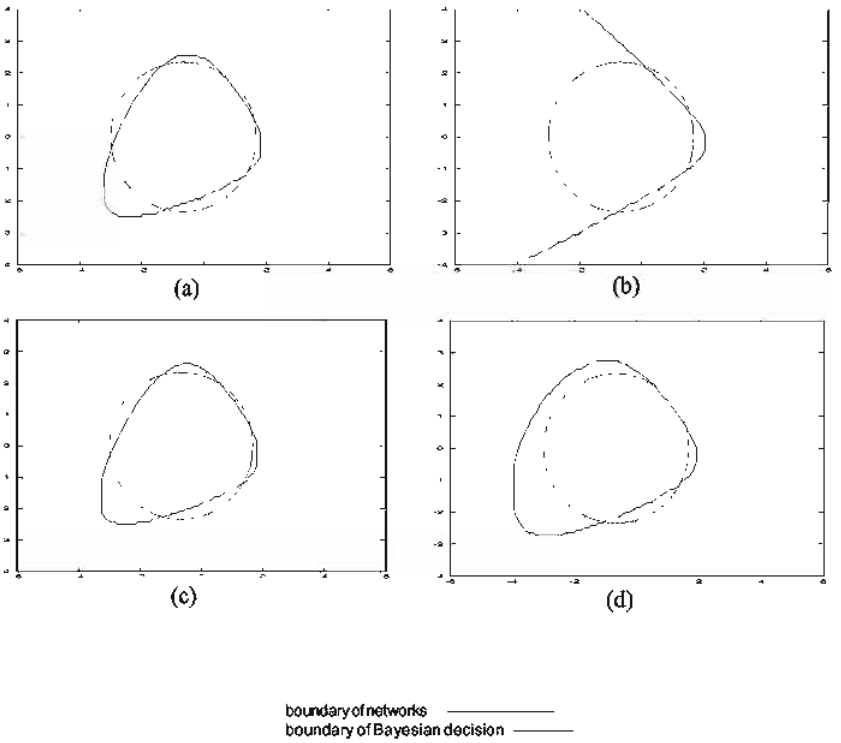


Figure 3: Decision boundaries formed by the different networks trained by the independent training (i.e., $\lambda = 0.0$ in negative correlation learning): (a) Network 1; (b) Network 2; (c) Network 3; (d) Ensemble; the circle represents the optimum Bayes solution



correlation learning and the independent training. In comparison of the average correct classification percentage and the decision boundaries obtained by the two ensemble learning methods, it is clear that negative correlation learning outperformed the independent training method. Although the classification performance of individual networks in the independent training is relatively good, the overall performance of the entire ensemble was not improved because different networks, such as Network 1 and Network 3 in Figure 3, tended to generate the similar decision boundaries.

The percentage of correct classification of the ensemble trained by negative correlation is 81.41, which is almost equal to that realised by the Bayesian classifier. Figure 2 clearly demonstrates that negative correlation learning is capable of constructing a decision between Class 1 and Class 2 that is almost as good as the optimum decision boundary. It is evident from Figure 2 that different individual networks trained by negative correlation learning were able to specialise to different parts of the testing set.

Table 4: Comparison between negativecorrelation learning (NCL) ($\lambda = 0.75$) and the independent training (i.e., $\lambda = 0.0$ in negative correlation learning) on the classification performance of individual networks and the ensemble; the results are the average correct classification percentage on the testing set over 10 independent runs

Methods	Net 1	Net 2	Net 3	Ensemble
NCL	81.11	75.26	73.09	81.03
Independent Training	81.13	80.49	81.13	80.99

ANALYSIS BASED ON THE CORRECT RESPONSE SETS

In this section, negative correlation learning was tested on the Australian credit card assessment problem. The problem is how to assess applications for credit cards based on a number of attributes. There are 690 patterns in total. The output has two classes. The 14 attributes include 6 numeric values and 8 discrete ones, the latter having from 2 to 14 possible values. The Australian credit card assessment problem is a classification problem which is different from the regression type of tasks, whose outputs are continuous. The data set was obtained from the UCI machine learning benchmark repository. It is available by anonymous ftp at ics.uci.edu (128.195.1.1) in directory /pub/machine-learning-databases.

Experimental Setup

The data set was partitioned into two sets: a training set and a testing set. The first 518 examples were used for the training set, and the remaining 172 examples for the testing set. The input attributes were rescaled to between 0.0 and 1.0 by a linear function. The output attributes of all the problems were encoded using a *1-of-m* output representation for m classes. The output with the highest activation designated the class. The aim of this section is to study the difference between negative correlation learning and independent training, rather than to compare negative correlation learning with previous work. The experiments used such a single train-and-test partition.

The ensemble architecture used in the experiments has 4 networks. Each individual network is a feedforward network with one hidden layer. Both hidden node function and output node function are defined by the logistic function in Eq.(18). All the individual networks have 10 hidden nodes. The number of training

epochs was set to 250. The strength parameter λ was set to 1.0. These parameters were chosen after limited preliminary experiments. They are not meant to be optimal.

Experimental Results

Table 5 shows the average results of negative correlation learning over 25 runs. Each run of negative correlation learning was from different initial weights. The ensemble with the same initial weight setup was also trained using BP without the correlation penalty terms (i.e., $\lambda = 0.0$ in negative correlation learning). Results are also shown in Table 5. For this problem, the simple averaging defined in Eq.(9) was first applied to decide the output of the ensemble. For the simple averaging, it was surprising that the results of negative correlation learning with $\lambda = 1.0$ were similar to those of independent training. This phenomenon seems contradictory to the claim that the effect of the correlation penalty term is to encourage different individual networks in an ensemble to learn different parts or aspects of the training data. In order to verify and quantify this claim, we compared the outputs of the individual networks trained with the correlation penalty terms to those of the individual networks trained without the correlation penalty terms.

Table 5: Comparison of error rates between negative correlation learning ($\lambda = 1.0$) and independent training (i.e., $\lambda = 0.0$ in negative correlation learning) on the Australian credit card assessment problem; the results were averaged over 25 runs. "Simple Averaging" and "Winner-Takes-All" indicate two different combination methods used in negative correlation learning, Mean, SD, Min and Max indicate the mean value,standard deviation, minimum and maximum value, respectively

	Error Rate	Simple Averageing	Winner-Takes-All
$\lambda = 1.0$	Mean	0.1337	0.1195
	SD	0.0068	0.0052
	Min	0.1163	0.1105
	Max	0.1454	0.1279
$\lambda = 0.0$	Mean	0.1368	0.1384
	SD	0.0048	0.0049
	Min	0.1279	0.1279
	Max	0.1454	0.1512

Table 6: The sizes of the correct response sets of individual networks created respectively by negative correlation learning (λ = 1.0) and independent training (i.e., λ = 0.0 in negative correlation learning) on the testing set and the sizes of their intersections for the Australian credit card assessment problem; the results were obtained from the first run among the 25 runs

$\lambda = 1.0$			$\lambda = 0.0$		
$\Omega_1 = 147$	$\Omega_2 = 143$	$\Omega_2 = 138$	$\Omega_1 = 149$	$\Omega_2 = 147$	$\Omega_2 = 148$
$\Omega_4 = 143$	$\Omega_{12} = 138$	$\Omega_{13} = 124$	$\Omega_4 = 148$	$\Omega_{12} = 147$	$\Omega_{13} = 147$
$\Omega_{14} = 141$	$\Omega_{23} = 116$	$\Omega_{24} = 133$	$\Omega_{14} = 147$	$\Omega_{23} = 147$	$\Omega_{24} = 146$
$\Omega_{34} = 123$	$\Omega_{123} = 115$	$\Omega_{124} = 133$	$\Omega_{34} = 146$	$\Omega_{123} = 147$	$\Omega_{124} = 146$
$\Omega_{134} = 121$	$\Omega_{234} = 113$	$\Omega_{1234} = 113$	$\Omega_{134} = 146$	$\Omega_{234} = 146$	$\Omega_{1234} = 146$

Two notions were introduced to analyse negative correlation learning. They are the correct response sets of individual networks and their intersections. The correct response set S_i of individual network *i* on the testing set consists of all the patterns in the testing set which are classified correctly by the individual network *i*. Let Ω_i denote the size of set S_i, and $\Omega_{i1i2\cdots ik}$ denote the size of set $S_{i1} \cap S_{i2} \cap \cdots \cap S_{ik}$. Table 6 shows the sizes of the correct response sets of individual networks and their intersections on the testing set, where the individual networks were respectively created by negative correlation learning and independent training. It is evident from Table 6 that different individual networks created by negative correlation learning were able to specialise to different parts of the testing set. For instance, in Table 6 the sizes of both correct response sets S_2 and S_4 at $\lambda = 1.0$ were 143, but the size of their intersection $S_2 \cap S_4$ was 133. The size of $S_1 \cap S_2 \cap S_3 \cap S_4$ was only 113. In contrast, the individual networks in the ensemble created by independent training were quite similar. The sizes of correct response sets S_1, S_2, S_3 and S_4 at $\lambda = 0.0$ were from 147 to 149, while the size of their intersection set $S_1 \cap S_2 \cap S_3 \cap S_4$ reached 146. There were only three different patterns correctly classified by the four individual networks in the ensemble.

In simple averaging, all the individual networks have the same combination weights and are treated equally. However, not all the networks are equally important. Because different individual networks created by negative correlation learning were able to specialise to different parts of the testing set, only the outputs of these specialists should be considered to make the final decision about the ensemble for this part of the testing set. In this experiment, a winner-takes-all method was applied to select such networks. For each pattern of the testing set,

the output of the ensemble was only decided by the network whose output had the highest activation. Table 5 shows the average results of negative correlation learning over 25 runs using the winner-takes-all combination method. The winner-takes-all combination method improved negative correlation learning significantly because there were good and poor networks for each pattern in the testing set, and winner-takes-all selected the best one. However, it did not improve the independent training much because the individual networks created by the independent training were all similar to each other.

CONCLUSIONS

This chapter describes negative correlation learning for designing neural network ensembles. It can be regarded as one way of decomposing a large problem into smaller and specialised ones, so that each sub-problem can be dealt with by an individual neural network relatively easily. A correlation penalty term in the error function was proposed to minimise mutual information and encourage the formation of specialists in the ensemble.

Negative correlation learning has been analysed in terms of mutual information on a regression task in the different noise conditions. Unlike independent training which creates larger mutual information among the ensemble, negative correlation learning can produce smaller mutual information among the ensemble. Through minimisation of mutual information, very competitive results have been produced by negative correlation learning in comparison with independent training.

This chapter compares the decision boundaries and the correct response sets constructed by negative correlation learning and the independent training for two pattern classification problems. The experimental results show that negative correlation learning has a very good classification performance. In fact, the decision boundary formed by negative correlation learning is nearly close to the optimum decision boundary generated by the Bayes classifier.

There are, however, some issues that need resolving. No special considerations were made in optimisation of the size of the ensemble and strength parameter λ in this chapter. Evolutionary ensembles with negative correlation learning for optimisation of the size of the ensemble had been studied on the classification problems (Liu, Yao & Higuchi, 2000).

REFERENCES

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Trans. Appl. Comp.*, AC-19, 716-723.

Clemen, R. T., & Winkler, R. L. (1985). Limits for the precision and value of information from dependent sources. *Operations Research*, 33:427-442.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y. & Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, 6:1289-1301.

Drucker, H., Schapire, R. & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In Hanson, S. J., Cowan, J. D. & Giles,C. L. (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 42-49. San Mateo, CA: Morgan Kaufmann.

Friedman, J. H. (1994). An overview of predictive learning and function approximation. In V. Cherkassky, J. H. Friedman, and H. Wechsler, (Eds.), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pp. 1-61. Springer-Verlag, Heidelberg, Germany.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(10):993-1001.

Jacobs, R. A. (1997). Bias/variance analyses of mixture-of-experts architectures. *Neural Computation*, 9:369-383.

Jacobs, R. A. & Jordan, M. I. (1991). A competitive modular connectionist architecture. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, (Eds.), *Advances in Neural Information Processing Systems 3*, pp. 767-773. Morgan Kaufmann, San Mateo, CA.

Jacobs, R. A., Jordan, M. I. & Barto, A. G. (1991). Task decomposition through competition in a modular connectionist architecture: the what and where vision task. *Cognitive Science*, 15:219-250.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J. & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79-87.

Jordan, M. I. & Jacobs, R. A. (1994). Hierarchical mixtures-of-experts and the em algorithm. *Neural Computation*, 6:181-214.

Liu, Y. & Yao, X. (1998a). Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems*, 4:176-185.

Liu, Y. & Yao, X. (1998b). A cooperative ensemble learning system. *In Proceedings of the 1998 IEEE International Joint Conference on Neural Networks (IJCNN'98)*, pages 2202-2207. IEEE Press, Piscataway, NJ, USA.

Liu, Y. & Yao, X. (1999). Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):716-725.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Trans. on Evolutionary Computation*, 4(4):380-725.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. New York: McGraw Hill.

Opitz, D. W. & Shavlik, J. W. (1996). Actively searching for an effective neural network ensemble. *Connection Science*, 8:337-353.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465-471.

Rosen, B. E. (1996). Ensemble learning using decorrelated neural networks. *Connection Science*, 8:373-383.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. I, pp. 318-362. MIT Press, Cambridge, MA.

Sarkar, D. (1996). Randomness in generalization ability: A source to improve it. *IEEE Trans. on Neural Networks*, 7(3):676-685.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:197-227.

Selfridge, O. G. (1958). Pandemonium: a paradigm for learning. *Mechanisation of Thought Processes: Proceedings of a Symp*. Held at the National Physical Lab., pp. 513-526. HMSO, London.

Sharkey, A. J. C. (1996). On combining artificial neural nets. *Connection Science*, 8:299-313.

van der Lubbe, J. C. A. (1997). *Information Theory*. Cambridge: Cambridge University Press.

van der Lubbe, J. C. A. (1999). *Information Theory*. (2nd ed) Prentice-Hall International, Inc.

Wallace, C. S., & Patrick, J. D. (1991). *Coding Decision Trees*. Technical Report 91/153, Department of Computer Science, Monash University, Clayton, Victoria 3168, Australia, August.

Wolpert, D. H. (1990). A mathematical theory of generalization. *Complex Systems*, 4:151–249.

Chapter II

A Perturbation Size-Independent Analysis of Robustness in Neural Networks by Randomized Algorithms

C. Alippi
Politecnico di Milano, Italy

ABSTRACT

This chapter presents a general methodology for evaluating the loss in performance of a generic neural network once its weights are affected by perturbations. Since weights represent the "knowledge space" of the neural model, the robustness analysis can be used to study the weights/performance relationship. The perturbation analysis, which is closely related to sensitivity issues, relaxes all assumptions made in the related literature, such as the small perturbation hypothesis, specific requirements on the distribution of perturbations and neural variables, the number of hidden units and a given neural structure. The methodology, based on Randomized Algorithms, allows reformulating the computationally intractable problem of robustness/sensitivity analysis in a probabilistic framework characterised by a polynomial time solution in the accuracy and confidence degrees.

INTRODUCTION

The evaluation of the effects induced by perturbations affecting a neural computation is relevant from the theoretical point of view and in developing an embedded device dedicated to a specific application.

In the first case, the interest is in obtaining a reliable and easy to be generated measure of the performance loss induced by perturbations affecting the weights of a neural network. The relevance of the analysis is obvious since weights characterise the "knowledge space" of the neural model and, hence, its inner nature. In this direction, a study of the evolution of the network's weights over training time allows for understanding the mechanism behind the generation of the knowledge space. Conversely, the analysis of a specific knowledge space (fixed configuration for weights) provides hints about the relationship between the weights space and the performance function. The latter aspect is of primary interest in recurrent neural networks where even small modifications of the weight values are critical to performance (e.g., think of the stability of an intelligent controller comprising a neural network and issues leading to robust control).

The second case is somehow strictly related to the first one and covers the situation where the neural network must be implemented in a physical device. The optimally trained neural network becomes the "golden unit" to be implemented within a finite precision representation environment as it happens in mission-critical applications and embedded systems. In these applications, behavioural perturbations affecting the weights of a neural network abstract uncertainties associated with the implementation process, such as finite precision representations (e.g., truncation or rounding in a digital hardware, fixed or low resolution floating point representations), fluctuations of the parameters representing the weights in analog solutions (e.g., associated with the production process of a physical component), ageing effects, or more complex and subtle uncertainties in mixed implementations.

The sensitivity/robustness issue has been widely addressed in the neural network community with a particular focus on specific neural topologies.

More in detail, when the neural network is composed of linear units, the analysis is straightforward and the relationship between perturbations and the induced performance loss can be obtained in a closed form (Alippi & Briozzo, 1998). Conversely, when the neural topology is non-linear, which is mostly the case, several authors assume the small perturbation hypothesis or particular hypothesis about the stochastic nature of the neural computation. In both cases, the assumptions make the mathematics more amenable with the positive consequence that a relationship between perturbations and performance loss can be derived (e.g., see Alippi & Briozzo, 1998; Pichè, 1995). Unfortunately, these analyses introduce hypotheses which are not always satisfied in all real applications.

Another classic approach requires expanding with Taylor the neural computation around the nominal value of the trained weights. A subsequent linear arised analysis follows which allows for solving the sensitivity issue (e.g., Pichè, 1995). Anyway, the validity of such approaches depend, in turn, on the validity of the small perturbation hypothesis: how to understand a priori if a perturbation is small for a given application?

In other applications the small perturbation hypothesis cannot be accepted being the involved perturbations everything but small. As an example we have the development of a digital embedded system. There, the designer has to reduce as possible the dimension of the weights by saving bits; this produces a positive impact on cost, memory size and power consumption of the final device.

Differently, other authors avoid the small perturbation assumption by focusing the attention on very specific neural network topologies and/or introducing particular assumptions regarding the distribution of perturbations, internal neural variables and inputs (Stevenson, Winter & Widrow, 1990; Alippi, Piuri & Sami, 1995).

Other authors have considered the sensitivity analysis under the small perturbation hypothesis to deal with implementation aspects. In this case, perturbations are specifically related to finite precision representations of the interim variables characterising the neural computation (Holt & Hwang, 1993; Dundar & Rose, 1995).

Differently from the limiting approaches provided in the literature, this chapter suggests a robustness/sensitivity analysis in the large, i.e., without assuming constraints on the size or nature of the perturbation; as such, small perturbation situations become only a subcase of the theory. The analysis is general and can be applied to all neural topologies, both static and recurrent in order to quantify the performance loss of the neural model when perturbations affect the model's weights.

The suggested sensitivity/robustness analysis can be applied to *All* neural network models involved in system identification, control signal/image processing and automation-based applications without any restriction. In particular, the analysis allows for solving the following problems:

- Quantify the robustness of a generically trained neural network by means of a suitable, easily to be computed and reliable robustness index;
- Compare different neural networks, solving a given application by ranking them according to their robustness;
- Investigate the criticality of a recurrent model ("stability" issue) by means of its robustness index;

- Study the efficacy and effectiveness of techniques developed to improve the robustness degree of a neural network by inspecting the improvement in robustness.

The key elements of the perturbation analysis are Randomised Algorithms – RAs- (Vidyasagar, 1996, 1998; Tempo & Dabbene, 1999; Alippi, 2002), which transform the computationally intractable problem of evaluating the robustness of a generic neural network with respect to generic, continuous perturbations, in a tractable problem solvable with a polynomial time algorithm by resorting to probability.

The increasing interest and the extensive use of Randomised Algorithms in control theory, and in particular in the robust control area (Djavan, Tulleken, Voetter, Verbruggen & Olsder, 1989; Battarcharyya, Chapellat & Keel, 1995; Bai & Tempo, 1997; Chen & Zhou, 1997; Vidyasagar, 1998; Tempo & Dabbene, 1999, Calafiore, Dabbene & Tempo, 1999), make this versatile technique extremely interesting also for the neural network researcher.

We suggest the interested reader to refer to Vidyasagar (1998) and Tempo and Dabbene (1999) for a deep analysis of the use of RAs in control applications; the author forecasts an increasing use of Randomised Algorithms in the analysis and synthesis of intelligent controllers in the neural network community.

The structure of the chapter is as follows. We first formalise the concept of robustness by identifying a natural and general index for robustness. Randomised Algorithms are then briefly introduced to provide a comprehensive analysis and adapted to estimate the robustness index. Experiments then follow to shed light on the use of the theory in identifying the robustness index for static and recurrent neural models.

A GENERAL ROBUSTNESS/SENSITIVITY ANALYSIS FOR NEURAL NETWORKS

In the following we consider a *generic* neural network implementing the $\hat{y} = f(\hat{\theta}, x)$ function where $\hat{\theta}$ is the weight (and biases) vector containing all the trained free parameters of the neural model.

In several neural models, and in particular in those related to system identification and control, the relationship between the inputs and the output of the system are captured by considering a regressor vector φ, which contains a limited time-window of actual and past inputs, outputs and possibly predicted outputs.

Of particular interest, in the zoo of neural models, are those which can be represented by means of the model structures $\hat{y}(t) = f(\varphi)$ where function $f(\cdot)$ is a regression-type neural network, characterised by N_φ inputs, N_h non-linear hidden units and a single effective linear/non-linear output (Ljung, 1987; Hertz, Krog & Palmer, 1991; Hassoun, 1995; Ljung, Sjoberg & Hjalmarsson, 1996).

The absence/presence of a dynamic in the system can be modelled by a suitable number of delay elements (or time lags), which may affect inputs (time history on external inputs u) system outputs (time history on $y(t)$) on predicted outputs (time history on $\hat{y}(t)$) or residuals (time history on $e(t) = \hat{y}(t) - y(t)$). Where it is needed $y(t)$, $\hat{y}(t)$ and $e(t)$ are vectorial entities, a component for each independent distinct variable.

Several neural model structures have been suggested in the literature, which basically differ in the regressor vector. Examples are, NARMAX and NOE topologies. NARMAX structure can be obtained by considering both past inputs and outputs of the system to infer $y(t)$. We have:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u), y(t-1), \cdots, y(t-n_y), \ldots, e(t-1), \cdots, e(t-n_e)\right]$$

Differently, the NOE structure processes only past inputs and predicted outputs, i.e.:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u), \hat{y}(t-1), \cdots, \hat{y}(t-n_y)\right].$$

Static neural networks, such as classifiers, can be obtained by simply considering external inputs:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u)\right].$$

Of course, different neural models can be considered, e.g., fully recurrent and well fit with the suggested robustness analysis.

A general, perturbation size independent, model-independent robustness analysis requires the evaluation of the loss in performance induced by a generic perturbation, in our analysis affecting the weights of a generic neural network. We denote by $y_\Delta(x) = f_\Delta(\theta, \Delta, x)$ the mathematical description of the perturbed computation and by $\Delta \in D \subseteq \Re^p$ a generic p-dimensional perturbation vector, a component for each independent perturbation affecting the neural computation $\hat{y}(t)$. The perturbation space D is characterised in stochastic terms by providing the probability density function pdf_D.

To measure the discrepancy between $y_\Delta(x)$ and $y(t)$ or $\hat{y}(t)$, we consider a generic loss function $U(\Delta)$. In the following we only assume that such performance loss function is measurable according to Lebesgue with respect to D. Lebesgue measurability for $U(\Delta)$ allows us for taking into account an extremely large class of loss functions.

Common examples for U are the Mean Square Error—MSE—loss functions

$$U(\Delta)=\frac{1}{N_x}\sum_{i=1}^{N_x}(\hat{y}(x_i)-\hat{y}(x_i,\Delta))^2 \text{ and } U(\Delta)=\frac{1}{N_x}\sum_{i=1}^{N_x}(y(x_i)-\hat{y}(x_i,\Delta))^2 . \quad (1)$$

More specifically, (1)-left compares the perturbed network with $\hat{y}$, which is supposed to be the "golden" error-free unit while (1)-right estimates the performance of the error-affected (perturbed) neural network (generalisation ability of the perturbed neural model).

The formalisation of the impact of perturbation on the performance function can be simply derived:

Definition: Robustness Index

We say that a neural network is robust at level $\bar{\gamma}$ in D, when the robustness index $\bar{\gamma}$ is the minimum positive value for which

$$U(\Delta) \le \bar{\gamma}, \forall \Delta \in D, \ \forall \gamma \ge \bar{\gamma} \quad (2)$$

Immediately, from the definition of robustness index we have that a generic neural network NN_1 is more robust than NN_2 if $\bar{\gamma}_1 < \bar{\gamma}_2$ and the property holds independently from the topology of the two neural networks.

The main problem related to the determination of the robustness index $\bar{\gamma}$ is that we have to compute $U(\Delta), \forall \Delta \in D$ if we wish a tight bound. The $\bar{\gamma}$ -identification problem is therefore intractable from a computational point of view if we relax all assumptions made in the literature as we do.

To deal with the computational aspect we associate a dual probabilistic problem to (2):

Robustness Index: Dual Problem We say that a neural network is robust at level $\bar{\gamma}$ in D with confidence η, when $\bar{\gamma}$ is the minimum positive value for which

$$\Pr(U(\Delta) \le \bar{\gamma}) \ge \eta \quad \text{holds} \quad \forall \Delta \in D, \ \forall \gamma \ge \bar{\gamma} \quad (3)$$

The probabilistic problem is weaker than the deterministic one since it tolerates the existence of a set of perturbations (whose measure according to Lebesgue is 1-η) for which $u(\Delta) > \bar{\gamma}$. In other words, not more than 100η % of perturbations $\Delta \in D$ will generate a loss in performance larger than $\bar{\gamma}$.

Probabilistic and deterministic problems are "close" to each other when we choose, as we do, $\eta = 1$. Note that $\bar{\gamma}$ depends only on the size of D and the neural network structure.

The non-linearity with respect to Δ and the lack of a priori assumptions regarding the neural network do not allow computing (2) in a closed form for the general perturbation case. The analysis, which would imply testing $U\Delta$ in correspondence with a continuous perturbation space, can be solved by resorting to probability according to the dual problem and by applying Randomised Algorithms to solve the robustness/sensitivity problem.

RANDOMIZED ALGORITHMS AND PERTURBATION ANALYSIS

In this paragraph we briefly review the theory behind Randomised Algorithms and adapt them to the robustness analysis problem.

In the following we denote by $p_\gamma = \Pr\{U(\Delta) \le \gamma\}$ the probability that the loss in performance associated with perturbations in D is below a given—but arbitrary—value γ.

Probability p_γ is unknown, cannot be computed in a close form for a generic U function and neural network topology, and its evaluation requires exploration of the whole perturbation space D.

The unknown probability p_γ can be estimated by sampling D with N independent and identically distributed samples Δ_i; extraction must be carried out according to the *pdf* of the perturbation.

For each sample Δ_i we then generate the triplet

$$\{\Delta_i, U(\Delta_i), I(\Delta_i)\}, i = 1, N \text{ where } I(\Delta_i) = \begin{cases} 1 \text{ if } U(\Delta_i) \le \gamma \\ 0 \text{ if } U(\Delta_i) > \gamma \end{cases} \quad (4)$$

The true probability p_γ can now simply be estimated as

$$\hat{p}_N = \frac{1}{N} \sum_{i=1}^{N} I(\Delta_i) \quad (5)$$

Of course, when N tends to infinity, $\hat{p}_N$ converges to p_γ. Conversely, on a finite data set of cardinality N, the discrepancy between $\hat{p}_N$ and p_γ exists and can be

simply measured as $|p_\gamma - \hat{p}_N|$. $|p_\gamma - \hat{p}_N|$ is a random variable which depends on the particular extraction of the N samples since different extractions of N samples from D will provide different estimates for $\hat{p}_N$. By introducing an accuracy degree ε on $|p_\gamma - \hat{p}_N|$ and a confidence level $1-\delta$ (which requests that the $|p_\gamma - \hat{p}_N| \le \varepsilon$ inequality is satisfied at least with probability $1-\delta$), our problem can be formalised by requiring that the inequality

$$\Pr\left\{|p_\gamma - \hat{p}_N| \le \varepsilon\right\} \ge 1 - \delta \tag{6}$$

is satisfied for $\forall \gamma \ge 0$. Of course, we wish to control the accuracy and the confidence degrees of (6) by allowing the user to choose the most appropriate values for the particular need. Finally, by extracting a number of samples from D according to the Chernoff inequality (Chernoff, 1952)

$$N \ge \frac{\ln \frac{2}{\delta}}{2\varepsilon^2} \tag{7}$$

we have that $\Pr\left\{|p_\gamma - \hat{p}_N| \le \varepsilon\right\} \ge 1 - \delta$ holds for $\forall \gamma \ge 0, \forall \delta, \varepsilon \in [0,1]$.

As an example, by considering 5% in accuracy and 99% in confidence, we have to extract 1060 samples from D; with such choice we can approximate p_γ with $\hat{p}_N$ introducing the maximum error 0.05 ($\hat{p}_N - 0.05 \le p_\gamma \le \hat{p}_N + 0.05$) and the inequality holds at least with the probability 0.99.

Other bounds can be considered instead of the Chernoff's one as suggested by Bernoulli and Bienaymè, (e.g., see Tempo & Dabbene, 1999). Nevertheless, the Chernoff's bound improves upon the others and, therefore, should be preferred if we wish to keep minimal the number of samples to be extracted. The Chernoff bound grants that:

- N is independent from the dimension of D (and hence it does not depend on the number of perturbations we are considering in the neural network);
- N is linear in $\ln \frac{1}{\delta}$ and $\frac{1}{\varepsilon^2}$ (hence it is polynomial in the accuracy and confidence degrees).

As a consequence, the dual probabilistic problem related to the identification of the robustness index $\bar{\gamma}$ can be solved with randomised algorithms and therefore with a polynomial complexity in the accuracy and the confidence degrees independently from the number of weights of the neural model network. In fact, by expanding the (6) we have that

$$\Pr\left\{p_\gamma - \hat{p}_N \le \varepsilon\right\} \ge 1-\delta \equiv \Pr\left\{\left|\Pr(u(\Delta)\le\gamma) - \frac{1}{N}\sum_i I(\Delta_i)\right| \le \varepsilon\right\} \ge 1-\delta \quad (8)$$

If accuracy ε and confidence δ are small enough, we can confuse p_γ and $\hat{p}_N$ by committing a small error. As a consequence, the dual probabilistic problem requiring $p_\gamma \ge \eta$ becomes $\hat{p}_N \ge \eta$. We surely assume ε and δ to be small enough in subsequent derivations.

The final algorithm, which allows for testing the robustness degree $\bar{\gamma}$ of a neural network, is:

1. *Select ε and δ sufficiently small to have enough accuracy and confidence.*
2. *Extract from D, according to its pdf, a number of perturbations N as suggested by (7).*
3. *Generate the indicator function $I(\Delta)$ and generate the estimate $\hat{p}_N = \hat{p}_N(\gamma)$ according to (5).*
4. *Select the minimum value γ_η from the $\hat{p}_N = \hat{p}_N(\gamma)$ function so that $\hat{p}_N(\gamma_\eta) = 1$ is satisfied $\forall \gamma \ge \gamma_\eta$. γ_η is the estimate of the robustness index $\bar{\gamma}$.*

Note that with a simple algorithm we are able to estimate in polynomial time the robustness degree $\bar{\gamma}$ of a generic neural network. The accuracy in estimating $\bar{\gamma}$ can be made arbitrarily good at the expense of a larger number of samples as suggested by Chernoff's bound.

APPLYING THE METHODOLOGY TO STUDY THE ROBUSTNESS OF NEURAL NETWORKS

In the experimental section we show how the robustness index for neural networks can be computed and how it can be used to characterise a neural model. After having presented and experimentally justified the theory supporting Randomised Algorithms, we will focus on the following problems:

- test the robustness of a given static neural network (robustness analysis);
- study the relationships between the robustness of a static neural network and the number of hidden units (structure redundancy);
- analyse the robustness of recurrent neural networks (robustness/stability analysis).

In the following experiments we consider perturbations affecting weights and biases of a neural network defined in D and subject to uniform distributions. Here,

a perturbation Δ_i affecting a generic weight w_i must be intended as a relative perturbation with respect to the weight magnitude according to the multiplicative perturbation model $w_{i,p} = w_i(1+\Delta_i), \forall i = 1, n$. A $t\%$ perturbation implies that Δ_i is drawn from a symmetrical uniform distribution of extremes

$$\left[-\frac{t}{100}, \frac{t}{100}\right];$$

a 5% perturbation affecting weights and biases composing vector $\hat{\theta}$ implies that each weight/bias is affected by an independent perturbation extracted from the [-0.05, 0.05] interval and applied to the nominal value according to the multiplicative perturbation model.

Experiment 1: *The impact of ε, δ and N on the evaluation of the robustness index*

The reference application to be learned is the simple error-free function

$$y = -x \cdot sin(x^2) + \frac{e^{-0.23 \cdot x}}{1+x^4}, \quad x \in [-3,3]$$

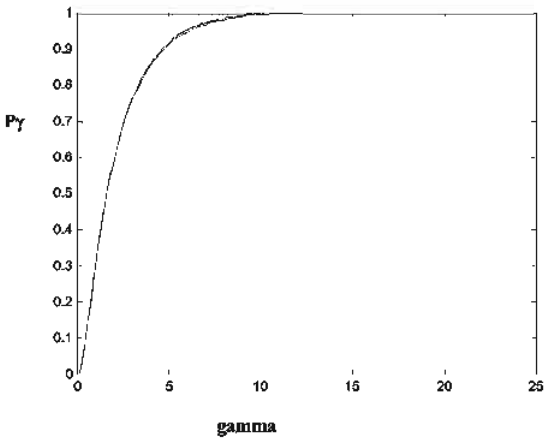
A set of 41 training data have been extracted from the function domain according to a uniform distribution. We considered static feedforward neural networks with hidden units characterised by a hyperbolic tangent activation function and a single linear output. Training was accomplished by considering a Levenberg-Marquardt algorithm applied to an MSE training function; a test set was considered during the training phase to determine the optimal stopping point so as to monitor the upsurgence of overfitting effects.

We discovered that all neural networks with at least 6 hidden units are able to solve the function approximation task with excellent performance.

In this experiment we focus the attention on the neural network characterised by 10 hidden units. After training we run the robustness algorithm by considering 7.5% of perturbations (weights are affected by perturbations up to 7.5% of their magnitude) and we chose $\varepsilon = 0.02$ and $\delta = 0.01$ from which we have to extract N=6624 samples from D. We carried out three extractions of N samples and, for each set, we computed the related $\hat{p}_N = \hat{p}_N(\gamma)$ curve.

The $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Figure 1. As we can see the curves are very close to each other. In fact, we know from the theory, that the estimated probability belongs to a neighbourhood of the true one according to the

Figure 1: $\hat{p}_N = \hat{p}_N(\gamma)$ for three different runs



$\hat{p}_N - 0.02 \le p_\gamma \le \hat{p}_N + 0.02$ relationship. A single curve is therefore enough to characterise the robustness of the envisaged neural network and there is no need to consider multiple runs. By inspecting Figure 1 we obtain that the estimate of the robustness index $\bar{\gamma}$ is γ_η=11 which implies that $U(\Delta) \le 11, \forall \Delta \in D$ with high probability.

We wish now to study the impact of N on $\hat{p}_N = \hat{p}_N(\gamma)$ by considering three runs with different ε and δ according to Table 1.

The $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Figure 2. It is interesting to note, at least for the specific application, that even with low values of N, the estimates for $\hat{p}_N = \hat{p}_N(\gamma)$ and γ_η are reasonable and not far from each other. We should anyway extract the number of samples according to Chernoff's inequality.

Experiment 2: *Testing the robustness of a given neural network*

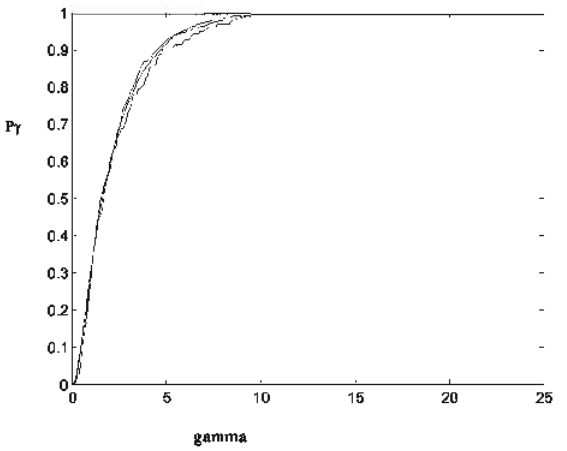
In the second experiment we test the robustness of the 10 hidden units network by considering its behaviour once affected by stronger perturbations (larger D) and, in particular, for perturbations 1%, 3%, 5%, 10%, 30%. We selected ε=0.02 and δ= 0.01.

The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ function corresponding to the different perturbations is given in Figure 3.

Table 1: ε, δ and N

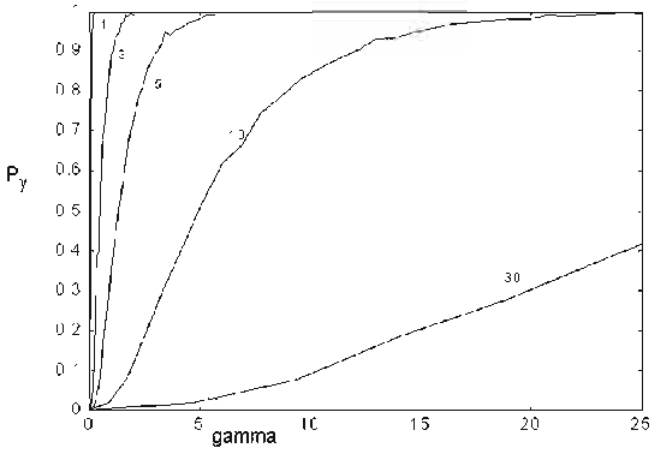
ε	δ	N
0.02	0.01	6624
0.05	0.05	738
0.1	0.1	150

Figure 2: $\hat{p}_N = \hat{p}_N(\gamma)$ for different runs with parameters given in Table 1



Again, from its definition, $\bar{\gamma}$ is the smallest value for which $\hat{p}_\gamma = 1$, $\gamma \geq \bar{\gamma}$; as an example, if we consider the 5% perturbation case, $\bar{\gamma}$ assumes a value around 7. It is obvious, but interesting to point out that, by increasing the strength of perturbation (i.e., by enlarging the extremes of the uniform distribution characterising the *pdf* of D), $\bar{\gamma}$ increases. In fact, stronger perturbations have a worse impact on the performance loss function since the error-affected neural network diverges from the error-free one. Conversely, we see that small perturbations, e.g., the 1% one, induce a very small loss in performance since the robustness index γ_η is very small.

Figure 3: $\hat{p}_\gamma$ as a function of γ for the 10 hidden units neural network



Experiment 3: *Testing the robustness of a hierarchy of performance-equivalent neural networks*

Once we have identified the robustness degree of a neural network solving an application, we can investigate whether it is possible to improve the robustness degree of the application by considering a sort of structural redundancy or not. This issue can be tackled by considering the neural hierarchy M: $M_1 \subseteq M_2 ... \subseteq M_k ...$ where M_k represents a neural network with k hidden units.

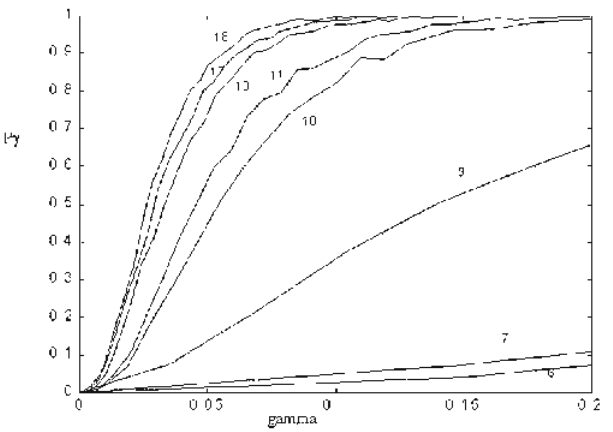
To this end, we consider a set of performance-equivalent neural networks, each of which is able to solve the application with a performance tolerable by the user. All neural networks are characterised by a different topological complexity (number of hidden units).

The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curves parameterised in the number of hidden units are given in Figure 4 in the case of 1% perturbation. We can see that by increasing the number of hidden units, $\bar{\gamma}$ decreases. We immediately realise that neural networks with a reduced number of hidden units are, for this application, less robust than the ones possessing more degrees of freedom. Large networks provide, in a way, a sort of spatial redundancy: information characterising the knowledge space of the neural networks is distributed over more degrees of freedom.

We discovered cases where a larger neural network was less robust than a smaller one: in such a case probably the complex model degenerates into a simpler one.

The evolution of $\bar{\gamma}$ over the number of hidden units parameterised with respect to the different perturbations 5%, 10% and 30% is given in Figure 5. We note that the minimal network, namely the smallest network able to solve the application, is not the more robust one for this application (in fact it possesses large values for the $\bar{\gamma}$ s).

Figure 4: $\hat{p}_\gamma$ *over* γ *and parameterised in the number of hidden units*



This trend—verified also with other applications—suggests that the robustness degree of the neural network improves on the average by increasing the number of hidden units (spatial redundancy). Anyway, with a small increase in the topological complexity (e.g., by considering the 13 hidden units model instead of the 6 one), we obtain a significant improvement according to the robustness level. There is no need to consider more complex neural networks since the improvement in robustness is small.

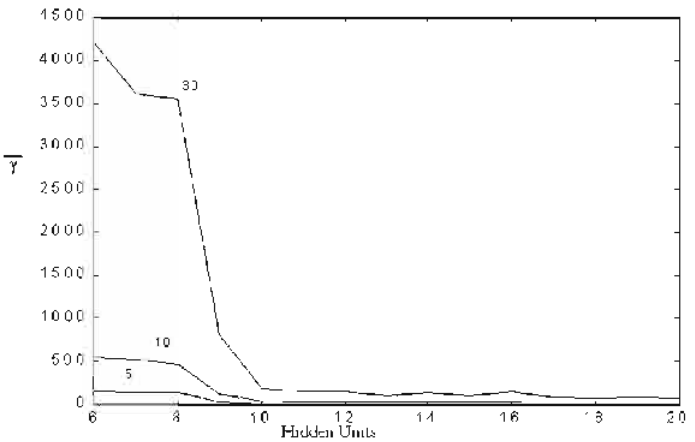
Experiment 4: *Testing the robustness of recurrent neural networks*

The goal of the last experiment is to study the robustness/stability of recurrent neural networks with the suggested theory. The chosen application refers to the identification of the open-loop stable, nonlinear, continuous system suggested in Norgaard (2000). The input and the corresponding output sequence of the system to be identified is given in Figure 6.

We first considered an NOE recurrent neural network with 5 hidden units characterised by the regressor vector $\varphi = [u(t-1), u(t-2), \hat{y}(t-1), \hat{y}(t-2)]$. The non-linear core of the neural network is a static regression type neural network as the one considered in the function approximation experiments. The topology of the NOE network is given in figure 7.

Once trained, the network we applied the methodology to estimates the robustness of the recurrent model. The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve, evaluated with $\varepsilon = \delta = 0.05$, for the 0.1% perturbation case is given in Figure 8. As we could have expected, differently from the static function approximation application, the recurrent NOE neural network is sensitive even to small perturbations affecting the knowledge space of the network.

Figure 5: $\bar{\gamma}$ *as function of the hidden units,* ε *=0.04,* δ *=0.01*



We identified the dynamic system with a NARMAX neural network characterised by 5 hidden units and the structure given in Figure 9. For such topology we selected the regressor vector

$$\varphi = [u(t-1), u(t-2), y(t-1), \cdot y(t-2), e(t-1), e(t-2)]$$

Figure 10 shows the $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve. It is interesting to note that the NARMAX neural network is less robust than the corresponding NOE model. The basic reason for such behaviour is due to the fact that the recurrent model does not receive directly as input the fed-back network output but only the residual *e*.

During training the NOE model must somehow learn more deeply the concept of stability since even small variations of weights associated with the training phase weights update would produce a trajectory diverging from the system output to be mimicked. This effect is due to the pure fed-back structure of the NOE model which

Figure 6: The input and the corresponding output of the dynamic system

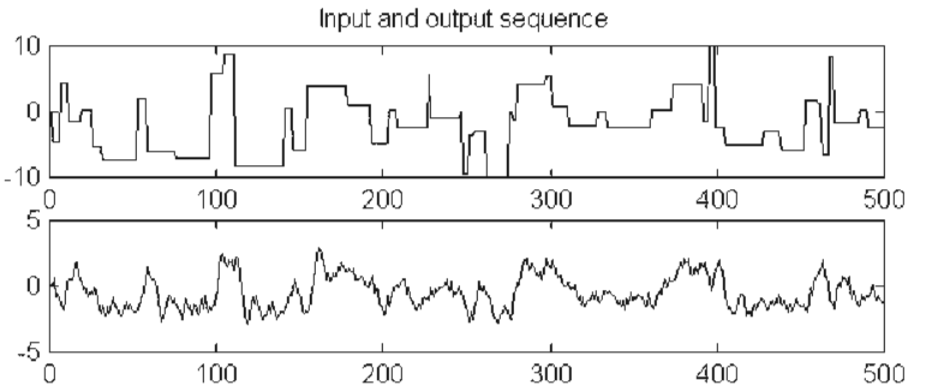


Figure 7: The considered NOE neural network

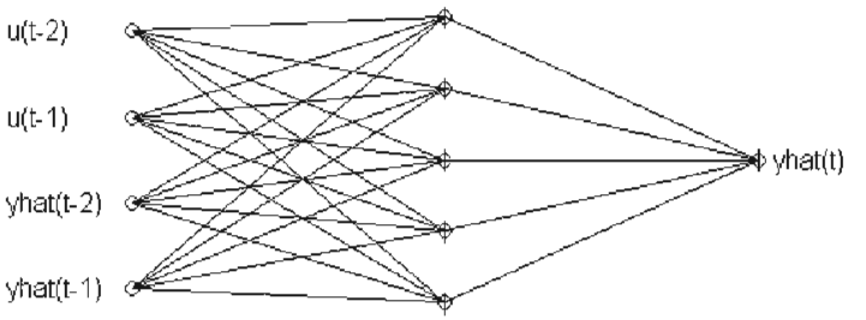


Figure 8: The $\hat{p}_\gamma$ function for the NOE neural network

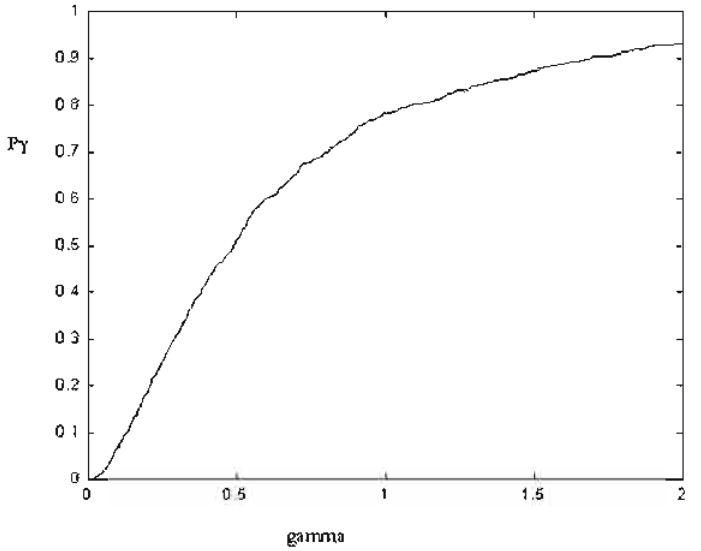
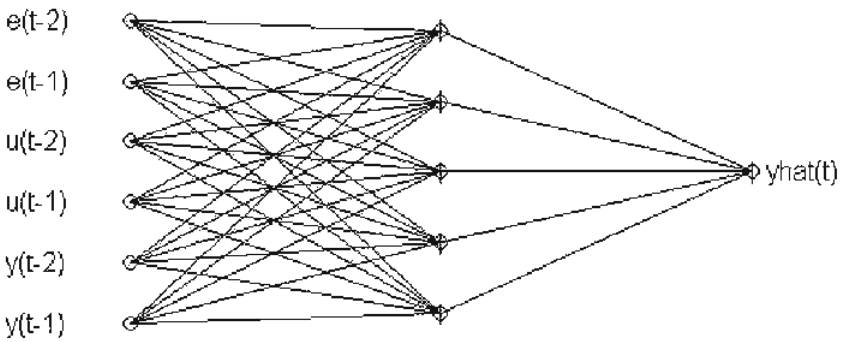
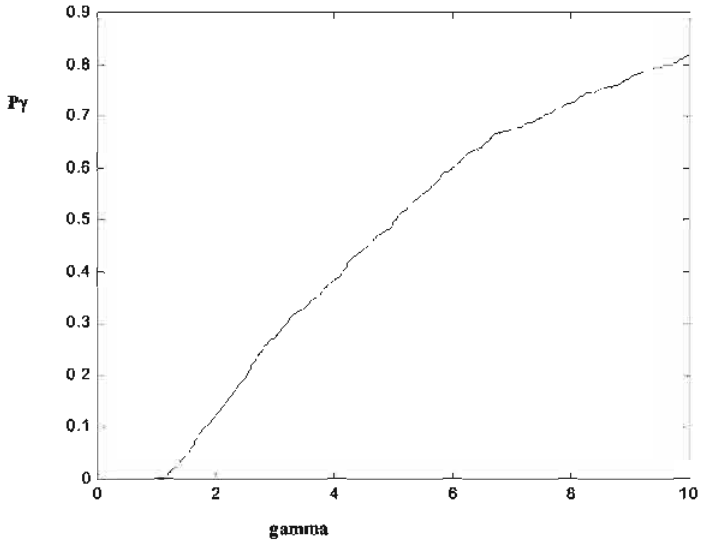


Figure 9: The considered NARMAX neural network



receives as inputs past predicted output and not direct information from the process. Interestingly, this requires the neural model to implicitly learn, during the training phase, the concept of robustness as proven by the $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve. Conversely, the NARMAX model has a smoother and less complex training phase since it receives fresh information directly from the process (y values) which help the neural model to be stable. As such, the training procedure will not search for weights configuration particularly robust since small deviations, which could make the system unstable, will be directly stabilised by the true information coming from the process.

Figure 10: The $\hat{p}_\gamma$ function for the NARMAX neural network



CONCLUSION

The main results of the chapter can be summarised as follows. Once given a trained neural network:

- the effects of perturbations affecting the network weights can be evaluated regardless of the topology and structure of the neural network, the strength of the perturbation by considering a probabilistic approach;
- the robustness/sensitivity analysis can be carried out with a Poly-time algorithm by resorting to Randomised Algorithms;
- the analysis is independent from the figure of merit considered to evaluate the loss in performance induced by the perturbations.

REFERENCES

Alippi, C. (2002). Randomized Algorithms: A system-level, Poly-time analysis of robust computation. *IEEE Transactions on Computers.* 51(7).

Alippi, C. & Briozzo, L. (1998). Accuracy vs. precision in digital VLSI architectures for signal processing. *IEEE Transactions on Computers,* 47(4).

Alippi, C., Piuri, V. & Sami, M. (1995). Sensitivity to Errors in Artificial Neural Networks: A Behavioural Approach. IEEE Transactions on Circuits and Systems: Part 1, 42(6).

Bai, E., Tempo, R. & Fu, M. (1997). Worst-case properties of the uniform distribution and randomized algorithms for robustness analysis. *IEEE-American Control Conference,* Albuquerque, NM.

Bhattacharyya, S.P, Chapellat, H. & Keel, L.H. (1995). *Robust Control: The Parametric Approach.* Englewoods Cliffs, NJ: Prentice Hall.

Calafiore, G., Dabbene, F. & Tempo, R. (1999). Uniform sample generation of vectors in lp balls for probabilistic robustness analysis. In *Recent Advances in Control,* Springer-Verlag.

Chen, X. & Zhou, K. (1997). On the probabilistic characterisation of model uncertainty and robustness. *Proceedings IEEE-36th Conference on Decision and Control,* San Diego, CA.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. 23.

Djavdan, P., Tulleken, H., Voetter, M., Verbruggen, H. & Olsder, G. (1989). Probabilistic Robust Controller Design. *Proceedings IEEE-28th Conference on Decision and Control,* Tampa, FL.

Dundar, G., Rose, K. (1995). The effects of Quantization on Multilayer Neural Networks, IEEE Transactions of Neural Networks. 6(6).

Hassoun, M.H. (1995). *Fundamentals of Artificial Neural Networks,* The MIT Press.

Hertz, J., Krogh, A. & Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation,* Addison-Wesley Publishing Co.

Holt, J. & Hwang, J. (1993). Finite precision error analysis of neural network hardware implementations. IEEE Transactions on Computers. 42(3).

Ljung, L. (1987). *System Identification, theory for the user,* Prentice-Hall.

Ljung, L. Sjoberg, J. & Hjalmarssoon, H. (1996). On neural networks model structures in system identification. In *Identification, Adaptation, Learning.* NATO ASI series.

Norgaard, M. (2000). Neural Network-Based System Identification Toolbox.

Piché, S. (1995). The selection of weights accuracies for Madalines. IEEE Transactions on Neural Networks. 6(2).

Stevenson, M., Winter, R. & Widrow, B. (1990). Sensitivity of Feedforward neural networks to weights errors, IEEE Transactions on Neural Networks. 1(1).

Tempo, R. & Dabbene, F. (1999). Probabilistic Robustness Analysis and Design of Uncertain Systems. Progress in Systems and Control Theory. 25.

Vidyasagar, M. (1996). *A Theory of Learning and Generalisation with Applications to Neural Networks and Control Systems.* Berlin: Springer-Verlag.

Vidyasagar, M. (1998). Statistical learning Theory and Randomized algorithms for Control. IEEE-Control systems.

Chapter III

Helicopter Motion Control Using a General Regression Neural Network

T.G.B. Amaral
Superior Technical School of Setúbal – IPS, Portugal

M.M. Crisóstomo
University of Coimbra, Portugal

V. Fernão Pires
Superior Technical School of Setúbal – IPS, Portugal

ABSTRACT

This chapter describes the application of a general regression neural network (GRNN) to control the flight of a helicopter. This GRNN is an adaptive network that provides estimates of continuous variables and is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. An important reason for using the GRNN as a controller is the fast learning capability and its non-iterative process. The disadvantage of this neural network is the amount of computation required to produce an estimate, which can become large if many training instances are gathered. To overcome this problem, it is described as a clustering algorithm to produce representative exemplars from a group of training instances that are close to one another reducing the computation amount to obtain an estimate. The reduction of training data used by the GRNN can make it possible to separate the obtained representative

exemplars, for example, in two data sets for the coarse and fine control. Experiments are performed to determine the degradation of the performance of the clustering algorithm with less training data. In the control flight system, data training is also reduced to obtain faster controllers, maintaining the desired performance.

INTRODUCTION

The application of a general regression neural network to control a non-linear system such as the flight of a helicopter at or near hover is described. This general regression neural network in an adaptive network that provides estimates of continuous variables and is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. The automatic flight control system, through the longitudinal and lateral cyclic, the collective and pedals are used to enable a helicopter to maintain its position fixed in space for a long period of time. In order to reduce the computation amount of the gathered data for training, and to obtain an estimate, a clustering algorithm was implemented. Simulation results are presented and the performance of the controller is analysed.

HELICOPTER MOTION CONTROL

Recently, unmanned helicopters, particularly large-scale ones, have been expected not only for the industrial fields such as agricultural spraying and aerial photography, but also for such fields as observation, rescuing and fire fighting. For monotonous and dangerous tasks, an autonomous flight control of the helicopter is advantageous.

In general, the unmanned helicopter is an example of an intelligent autonomous agent. Autonomous flight control involves some difficulties due to the following:

- it is non-linear;
- flight modes are cross-coupled;
- its dynamics are unstable;
- it is a multivariate (i.e., there are many input-output variables) system;
- it is sensitive to external disturbances and environmental conditions such as wind, temperature, etc;
- it can be used in many different flight modes (e.g., hover or forward flight), each of which requires different control laws;
- it is often used in dangerous environments (e.g., at low altitudes near obstacles).

These characteristics make the conventional control difficult and create a challenge to the design of intelligent control systems.

For example, although helicopters are non-linear systems, NN controllers are capable of controlling them because they are also inherently non-linear. The instabilities that result from time delays between changes in the system input and output can be addressed with the previous learning of the network with a set of data that represents the pilots knowledge to stabilize the helicopter. Linear NN can be implemented to compensate the cross-couplings between control inputs, mainly when the helicopter makes a significant change in its flight.

Therefore, a supervised general regression neural network can be used to control the flight modes of an unmanned helicopter. The regression is the least-mean-squares estimation of the value of a variable based on data samples. The term *general regression* implies that the regression surface is not restricted by being linear. If the values of the variables to be estimated are future values, the general regression network (GRNN) is a predictor. If they are dependent variables related to input variables in a process, system or plant, the GRNN can be used to model the process, system or plant. Once the system is modelled, a control surface can be defined in terms of samples of control variables that, given a state vector of the system, improve the output of the system. If a GRNN is trained using these samples, it can estimate the entire control surface, becoming a controller. A GRNN can be used to map from one set of sample points to another. If the target space has the same dimension as the input space, and if the mapping is one-to-one, an inverse mapping can easily be formed using the same examples. When the variables to be estimated are for intermediate values between given points, then the GRNN can be used as an interpolator.

In all cases, the GRNN instantly adapts to new data points. This could be a particular advantage for training robots to emulate a teacher or for any system whose model changes frequently.

SYSTEM MODELLING

The helicopter control is one of the popular non-linear educational control problems. Due to its highly non-linear dynamics, it gives the possibility to demonstrate basic features and limits of non-linear control concepts. Sugeno (1997, 1998) developed a fuzzy-logic based control system to replace the aircraft's normal set of control inputs. Other researchers, such has Phillips et al. (1994), Wade et al (1994), and Wade and Walker (1994), have developed fuzzy logic flight controls describing systems that include mechanisms for discovering and tuning fuzzy rules in adaptive controllers. (Larkin, 1984) described a model of an autopilot controller based on fuzzy algorithms. An alternative approach to real-time control of an

autonomous flying vehicle based on behavioral, or reactive, approach is proposed by Fagg et al. (1993). A recurrent neural network used to forward modeling of helicopter flight dynamics was described by Walker and Mo (1994). The NN-based controllers can indirectly model human cognitive performance by emulating the biological processes underlying human skill acquisition.

The main difference between NN-based controllers and conventional control systems is that, in the NN case, systems are built from indirectly representations of control knowledge similar to those employed by skilled humans, while in the conventional design case, a deep analytical understanding of the system dynamics is needed. The ability of humans to pilot manned helicopters with only the qualitative knowledge indicate that NN-based controllers with similar capabilities can also be developed.

The helicopter can be modelled as a linear system around trim points, i.e., a flight with no accelerations and no moments. The state space equations are a natural form, which can represent the helicopter motion. The general mathematical model is given by:

$$\dot{x} = Ax + Bu_c$$

$$y = Cx + Du_c$$

where x, u_c *and* y are the state vector, control vector and output vector, respectively.

The helicopter used to simulate the flight in hover position was a single main rotor helicopter of 15,000 pounds. The control and state vectors are defined as:

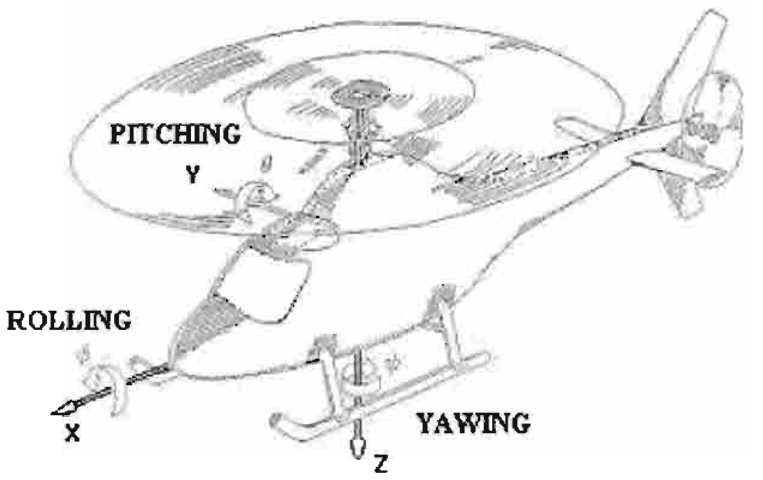
$$u_c^T = [\delta_a \ \delta_b \ \delta_c \ \delta_d] \tag{1}$$

$$x^T = [u \ v \ w \ p \ q \ r \ \phi \ \theta \ \varphi \ \mathrm{x} \ y \ z] \tag{2}$$

where

δ_a is the collective control [*inches*];
δ_b and δ_c are the longitudinal and lateral cyclic controls, respectively [*inches*];

Figure 1: Helicopter coordinates



δ_d is the pedal control [*inches*];
u, v and w are the perturbation linear velocities [*ft*/sec];
p, q and r are the perturbation angular velocities [*rad*/sec];
ϕ, θ and φ are the perturbation euler angles for roll, pitch and yaw [*rad*];
x, y and z are the perturbation linear displacements over the ground [*ft*].

Figure 1 shows the coordinate system to describe the motion of the helicopter. The origin of the helicopter axes is placed on the center of gravity.

The thrust of the main rotor, thus mainly the vertical acceleration, is controlled by the collective control (δ_a). The pitching moment, that is, nose pointing up or down, is controlled by the longitudinal cyclic control (δ_b). The rolling moment, that is, right wing tip down, left wing tip up, and vice versa, is controlled by the lateral cyclic control (δ_c). The yawing moment, that is, nose left and right, is controlled by the pedal control (δ_d).

The corresponding differential equations that represent the behavior of the helicopter in hover position are:

$$\frac{du}{dt} = -0.069u - 0.032v + 116.8p + 1168.5q - 6.15r - 32.19\theta + 0.118\delta_a$$
$$- 2.79\delta_b - 0.25\delta_c + 0.0043\delta_d$$

$$\frac{dv}{dt} = 0.017u - 0.085v - 0.0021w - 430.5p + 381.3q + 30.75r + 32.14\phi + 0.023\theta - 0.14\delta_a - \delta_b + 0.665\delta_c - 1.39\delta_d$$

$$\frac{dw}{dt} = -0.0021v - 0.257w + 7.99p + 46.74q + 135.3r + 1.85\phi - 0.404\theta - 9.23\delta_a - 0.107\delta_b - 0.01\delta_c$$

$$\frac{dp}{dt} = 0.45u - 0.687v - 0.0021w - 6027.2p + 5043.16q + 664.2r - 1.82\delta_a - 13.7\delta_b + 8.58\delta_c - 5.15\delta_d$$

$$\frac{dq}{dt} = 0.665u + 0.429v - 0.043w - 1537.5p - 15744.5q - 12.3r - 0.966\delta_a + 37.13\delta_b + 3.43\delta_c + 0.75\delta_d$$

$$\frac{dr}{dt} = -0.0214u + 0.515v + 0.0064w - 369.0p - 44.28q - 1266.9r + 25.97\delta_a - 0.15\delta_b + 0.075\delta_c + 40.78\delta_d$$

$$\frac{d\phi}{dt} = p$$

$$\frac{d\theta}{dt} = q$$

$$\frac{d\varphi}{dt} = r$$

$$\frac{dx}{dt} = u$$

$$\frac{dy}{dt} = v$$

$$\frac{dz}{dt} = w$$

Since each motion is not independent of δ_a, δ_b, δ_c and δ_d, there exists a cross-coupling.

Figure 2 shows the root locus for the model described above. Figure 2(a) shows the root locus, considering the collective control as the input and the vertical displacement as the output. In Figure 2(c), the longitudinal cyclic and the forward displacement are the input and the output, respectively. Figure 2(e) shows the root locus considering the lateral cyclic as the input and the lateral displacement as the output. Figures 2(b), (d) and (f) show the zoom of the region near the imaginary axis as well as the roots that dominate the transient response. In general, the contribution

Figure 2: Root locus of the helicopter model

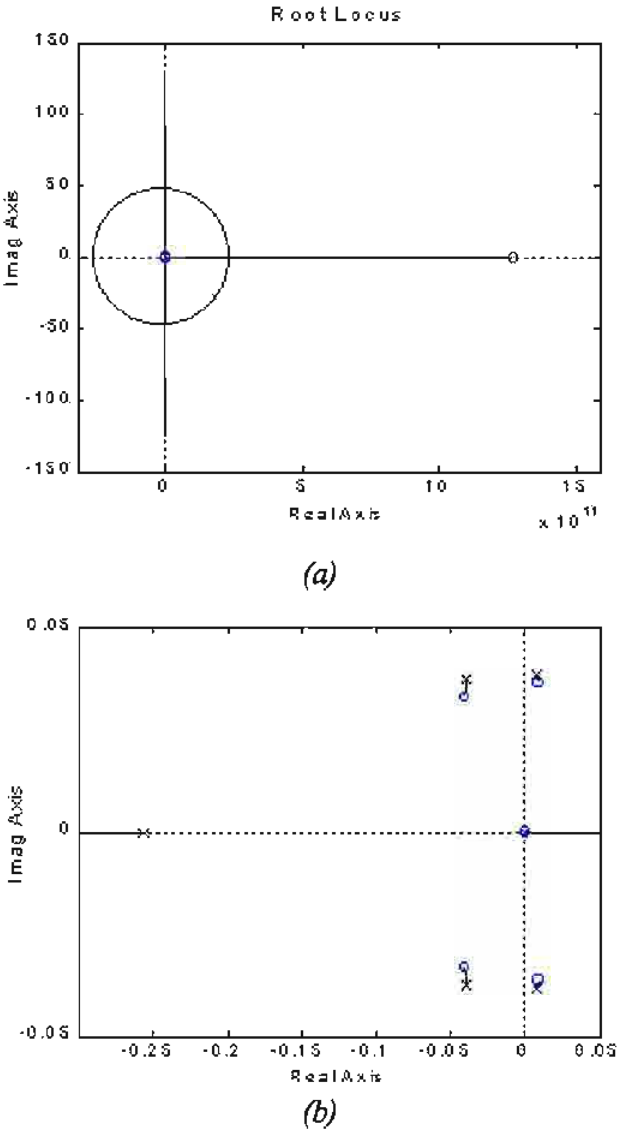
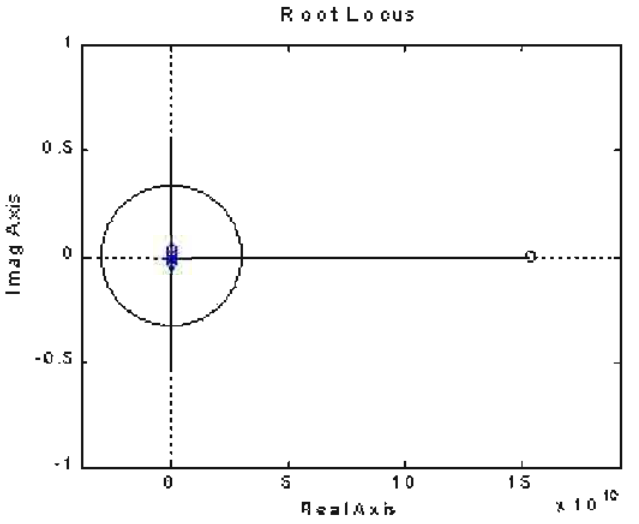
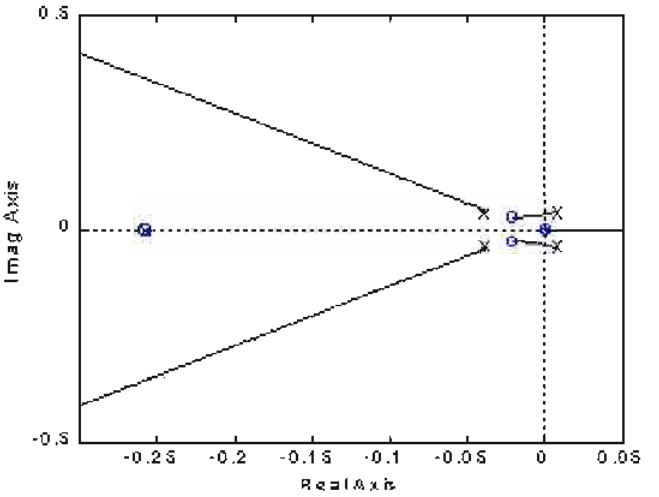


Figure 2: Root locus of the helicopter model (continued)



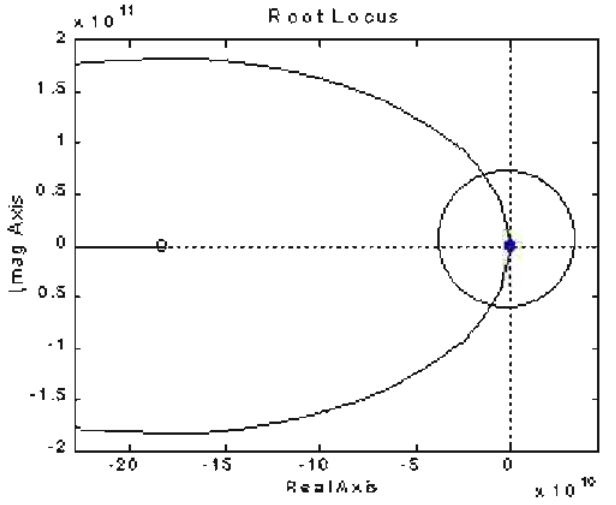
(c)



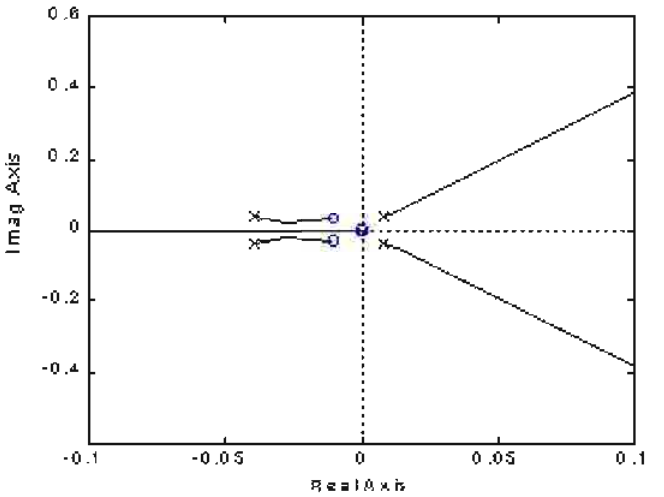
(d)

in the time response of roots that lie relatively far to the left in the s-plane will be small. These three Figures clearly show that some of the eigenvalues corresponding to the helicopter model are in the right side of the s-plane, with positive real-part values, making the system unstable.

Figure 2: Root locus of the helicopter model (continued)



(e)



(f)

GENERAL REGRESSION NEURAL NETWORK

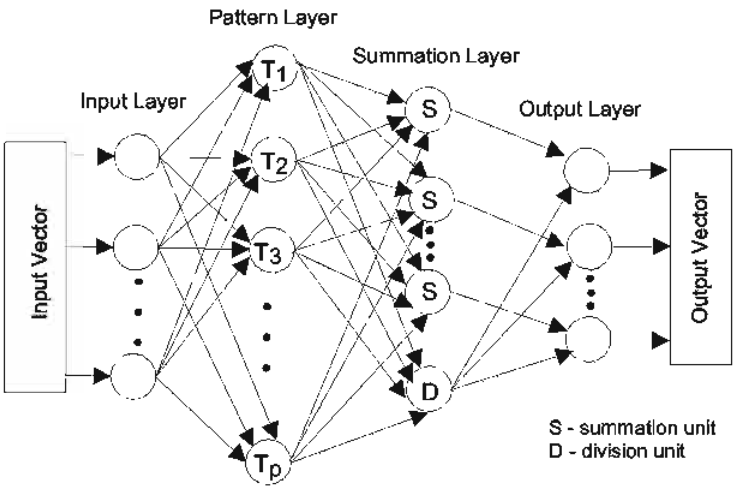
The generalized regression neural networks are memory-based feed-forward networks originally developed in the statistics literature by Nadaraya (1964) and known as Nadaraya-Watson kernel regression. Then the GRNN was 're-discovered' by Specht (1991) and Chen, C. (1996), with the desired capability of

learning in a one-shot manner from incoming training data being independent upon time-consuming iterative algorithms. This quick learning ability allows the GRNN to adapt to changing system parameters more rapidly than other methods such as genetic algorithms, back-propagation or reinforcement learning. This is achieved by the estimation of continuous variables with a single training instance and refining this estimation in a non-iterative manner since the training data is added to the network. Therefore, this neural network can be used as an intelligent controller for the autonomous helicopter.

GRNN Architecture

The GRNN is a special extension of the radial basis function network. This neural network is based on nonlinear regression theory consisting of four layers: the input layer, the pattern layer, the summation layer and the output layer (see Figure 3). It can approximate any arbitrary mapping between the input and output vectors. While the neurons in the first three layers are fully connected, each output neuron is connected only to some processing units in the summation layer. The summation layer has two different types of processing units: the summation units and the division unit. The number of summation units in the summation layer is always the same as the number of GRNN output units. The division unit only sums the weighted activations of the pattern units without using any activation function. Each of the output units is connected only to its corresponding summation unit and to the division unit. There are no weights in these connections. The function of the output units consists of a simple division of the signal coming from the summation unit by the signal coming from the division unit.

Figure 3: Topology of the generalized regression neural network



Consider X and Y independent and dependent variables respectively. The regression of Y on X is the computation of the most probable value of Y for each value of X based on a finite number of possibly noisy measurements of X and the associated values of Y. The variables X and Y can be vectors. In parametric regression, some functional form with unknown parameters, a_i, is assumed and the values of the parameters are chosen to make the best fit to the observed data. For example, in linear regression, the output Y is assumed to be a linear function of the input X, and the unknown parameters a_i are linear coefficients. In nonparametric regression, no assumption about the statistical structure of incoming training data is made.

The equation form used for the GRNN is presented in (3) and (4) (Specht, 1991). The resulting regression, which involves summations over the observations, is directly applicable to problems involving numerical data.

$$D_i^2 = (X - X_i)^T (X - X_i) \tag{3}$$

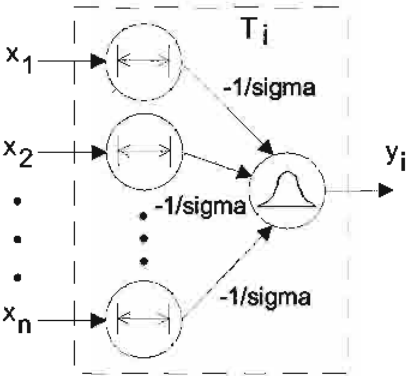
$$\hat{Y}(X) = \frac{\sum_{i=1}^{n} Y_i \exp\left(\frac{-D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{n} \exp\left(\frac{-D_i^2}{2\sigma^2}\right)} \tag{4}$$

X_i and Y_i are earlier samples that compose the training data set and n represents the number of training data.

The estimated output $\hat{Y}(X)$ is a weighted average of all the observed values Y_i, where each observed value is weighted exponentially according to the Euclidean distance between X and X_i. The smoothing parameter σ (or bandwidth) controls how tightly the estimate is made to fit the data. Figure 4 shows the shapes of the i^{th} pattern unit node in the pattern layer T_i of the GRNN for three different values of the smoothing parameter. The output value in each pattern unit T_i is given by the following expression:

$$T_i = \exp\left(-\|X - X_i\|^2 / 2\sigma^2\right) \tag{5}$$

Figure 4: Structure of the i^{th} pattern unit



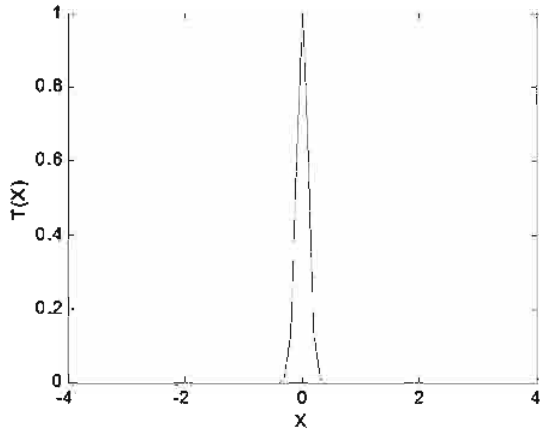
where $T_i : \Re^n \rightarrow \Re$ and $i = 1,..., p$ represents all the pattern units in the pattern layer.

When the smoothing parameter σ is made large, the estimate is forced to be smooth and in the limit becomes a multivariate Gaussian with covariance $\sigma^2 I$. On the other hand, a smaller σ allows the estimate to more closely fit the data and assume non-Gaussian shapes (see Figure 5(a)). In this case, the disadvantage happens when the wild points could have a great effect on the estimate. As σ becomes large, $\hat{Y}(X)$ assumes the value of the sample mean of the observed Y_i (see Figure 5(c)). When σ goes to 0, $\hat{Y}(X)$ becomes the value of the Y_i associated with the data closest to X. For intermediate values of σ, all values of Y_i are used, but those corresponding to observed values closer to X are given heavier weight (see Figure 5(b)).

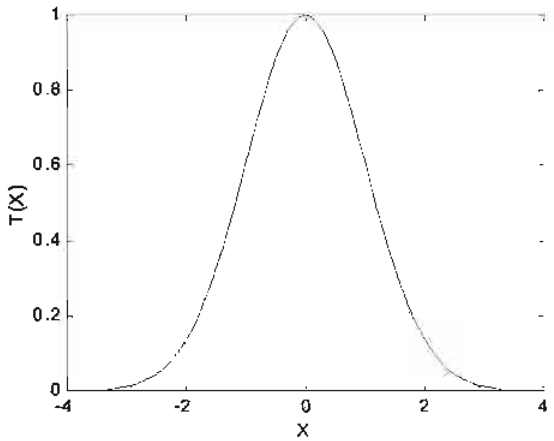
If we have much confidence in the input data, a small σ can be used to give greater weight to individual measurements. However, if there is uncertainty in the input data due to noise, a large σ must be used to reduce the effect of spurious measurements on an estimate. The optimisation of the smoothing parameter is critical to the performance of the GRNN. Usually this parameter is chosen by the cross-validation procedure or by esoteric methods that are not well known in the neural net literature.

In the next subsections it will be shown the application of the GRNN to model a piecewise linear function, and to control an unmanned helicopter. It is also discussed the clustering algorithm to obtain the representative samples between the training data (Lefteri, 1997).

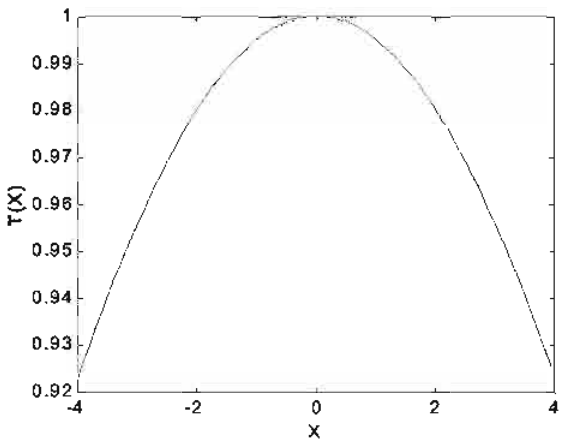
Figure 5: Possible shapes for different smoothing parameter values



(a) $\sigma = 0.1$



(b) $\sigma = 1$



(c) $\sigma = 10$

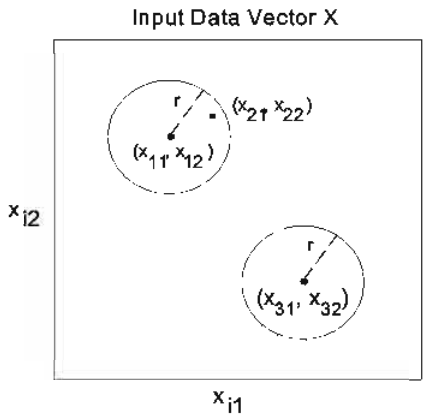
Clustering and Adaptation to Nonstationary Statistics

For some problems, the number of sample points (X, Y) may be not sufficient since it is desired to use all the data obtainable directly in the estimator (4). In other problems, the number of sample points can be sufficiently large, becoming no longer practical to assign a separate node to each sample. There exist various clustering techniques that can be used to group samples. Therefore the group can be represented by only one node (Moody & Darken, 1989), (Burrascano, 1991), and (Tou & Gonzalez, 1974). A sample point belongs to a group if the distance between this sample point and the cluster center is less than a specific value. This value, which can be considered the radius of influence r of the cluster, must need to be specified before the training starts.

In the developed clustering algorithm, representative samples are produced from a group of training instances that are close to one another. The training is completed after presenting to the GRNN input layer, only once, each input-output vector pair from the training set. The Euclidean distance to obtain the representative samples and to reject all the other data points was used in the algorithm.

Suppose that we have a set of n data samples $\{(X_i, Y_i) \in (X, Y); i = 1, \ldots, n\}$ where X and Y represent the input and output data sets, respectively. X_i and Y_i are a 2D vector (x_{i1}, x_{i2}) and a single value, respectively. Initially, the first sample point (X_1, Y_1) in the training set becomes the center of the cluster of the first pattern unit at X. The next sample point is then compared with this center of the first pattern unit, and it is assigned to the same cluster (pattern unit) if its distance from this center is less than the prespecified radius of influence. Then, equation (7) should be updated for this cluster. Otherwise, if the distance $|X - X_i|$ is higher than r, then the sample becomes the center of the cluster of the next pattern unit. Figure 6 shows how the clustering algorithm works considering three points (x_{11}, x_{12}), (x_{21}, x_{22}) and $(x_{31},$

Figure 6: Cluster generation



x_{32}). The first two points belong to the same cluster because the distance between them is less than r. The third point is the center of the new cluster since the distance is higher than r.

In the same manner, all the other sample points are compared one-by-one with all pattern units already set, and the whole pattern layer is thus gradually built. During this training, the determined values of individual elements of the center clusters are directly assigned to the weights in connections between the input units and the corresponding pattern units.

After the determination of the cluster centers, the equation (4) can then be rewritten as (Specht, 1991):

$$\hat{Y}(X) = \frac{\sum_{i=1}^{p} A_i \exp\left(\frac{-D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{p} B_i \exp\left(\frac{-D_i^2}{2\sigma^2}\right)} \tag{6}$$

where

$$\begin{cases} A_i(k) = A_i(k-1) + Y_j \\ B_i(k) = B_i(k-1) + 1 \end{cases} \tag{7}$$

The value $p \langle n$ represents the number of clusters. $A_i(k)$ and $B_i(k)$ are the coefficients for the cluster i after k samples. $A_i(k)$ is the sum of the Y values and $B_i(k)$ is the number of samples assigned to cluster i. The $A_i(k)$ and $B_i(k)$ coefficients are completely determined in one iteration for each data sample.

Reducing the Number of Clusters in Dynamic Systems

If the network is used to model a system with changing characteristics, it is necessary to eliminate the clusters that were not updated during a period of time. The justification for this is that in the dynamic systems appears new cluster centers that represent the new behavior of the model. Then, the number of clusters will increase and also the computation time to produce an output. Since the A and B coefficients can be determined by using the recursive equations (7), it is introduced a forgetting function allowing to reduce the number of clusters as shown in the following expressions,

$$\begin{cases} A_i(k) = A_i(k-1)\exp\left(-\frac{t_i}{\tau}\right) + \left(1-\exp\left(-\frac{t_i}{\tau}\right)\right)Y_j \\ B_i(k) = B_i(k-1)\exp\left(-\frac{t_i}{\tau}\right) + \left(1-\exp\left(-\frac{t_i}{\tau}\right)\right) \end{cases} \quad (8)$$

and

$$\begin{cases} A_i(k) = A_i(k-1)\exp\left(-\frac{t_i}{\tau}\right) \\ B_i(k) = B_i(k-1)\exp\left(-\frac{t_i}{\tau}\right) \end{cases} \quad (9)$$

Equation (8) is the update expression when a new sample is assigned to the cluster *i*. Equation (9) is applied to all other clusters. The parameters *t* and τ are the time passed after the last update of the cluster *i* and a constant that determines when the cluster disappears after the last update, respectively. Figure 7 shows the exponential decay and increase functions represented by solid and dashed lines respectively. The exponential decay function will attenuate all the coefficients *A* and *B* of the clusters. The increasing exponential function allows the new sample data to have an influence in the local area around its assigned cluster center.

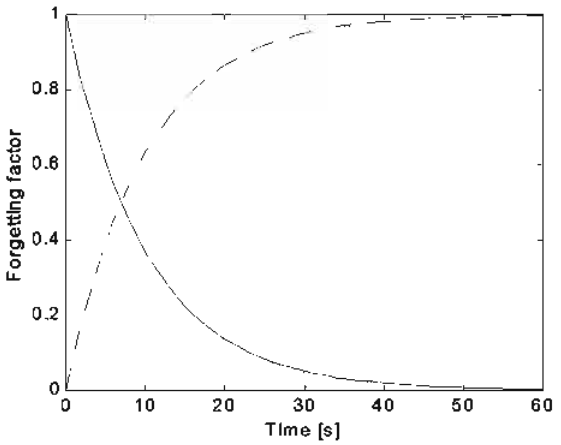
When the coefficient *B* is zero then the corresponding cluster would be eliminated. For example, considering Figure 7, if the cluster *i* is not updated during 60 seconds then the cluster *i* (and its associated A_i and B_i coefficients) will be eliminated.

Comparison with other Non-Linear Regression Techniques

The advantages of GRNN relative to other non-linear regression techniques are:

1. The network learns in one pass through the data and can generalize from samples as soon as they are stored.
2. With the increasing number of observed samples, the estimates converge to the conditional mean regression surfaces. However, using a few number of samples, it forms very reasonable regression surfaces.

Figure 7: Exponential decay function



3. The estimate is limited within a range defined by the minimum and maximum of the observations.
4. The estimate cannot converge to poor solutions corresponding to local minima of the error criterion.
5. A software simulation is easy to develop and to use.
6. The network can provide a mapping from one set of sample points to another. If the mapping is one-to-one, an inverse mapping can easily be generated from the same sample points.
7. The clustering version of GRNN, equation (6), limits the numbers of nodes. Optionally it can provide a mechanism for forgetting old data.

The main disadvantage of GRNN is the amount of computation required to produce an estimate, since it can become large if many training instances are gathered. To overcome this problem it was implemented a clustering algorithm. This additional processing stage brings the questions regarding when to do the initial clustering and if the re-clustering should be done after additional training data is gathered.

GRNN-Based Model

The described GRNN type has many potential uses as models and inverse models (see Figure 8). A simple problem with one independent variable is used as an example to show how the regression technique is applied to modelling a system. Suppose that we have a piecewise linear function and training instances taken from this function (see Figure 9) (Montgomery, 1999). The samples X_i = [–4,–3,–2,–1,0,1,2,3,4] and Y_i = [–1,–1,–1,–1,–1,0,1,1,1] are represented by circles.

Since GRNN always estimates using a weighted average of the given samples, the estimate is always within the observed range of the dependent variable. In the input range, the estimator takes on a set of curves that depend on σ, each of which is a reasonable approximation to the piecewise linear function (see Figure 9). Figure 10 shows the GRNN estimates of this function for different sigma values. For σ = 0.5 the curve is the best approximation. A small sigma allows the estimate to more closely fit the training data, while a large sigma produces a smoother estimate. It is possible to over fit the data with very small values of σ.

Besides the advantages of the GRNN when compared with other non-linear regression techniques, there exists another four benefits from the use of the GRNN. First, it has a non-iterative, fast-learning capability. Second, the smoothing parameter, σ, can be made large to smooth out noisy data or made small to allow the estimated regression surface to be as non-linear as required to more closely approximate the actual observed training data. Third, it is not necessary to specify the form of a regression equation. Finally, the addition of new samples to the training data set does not require re-calibrating the model. The disadvantage of the network is the

Figure 8: Modelling the system using GRNN

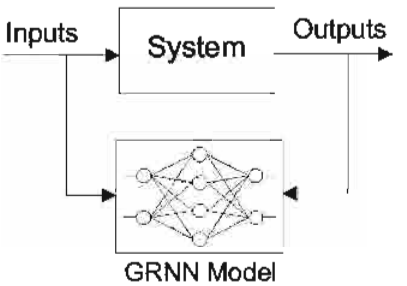


Figure 9: Linear function with training set

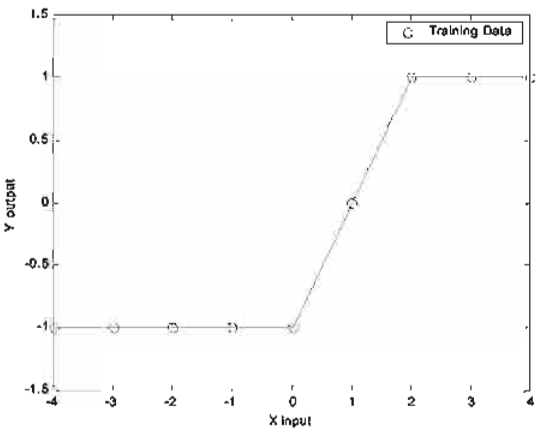
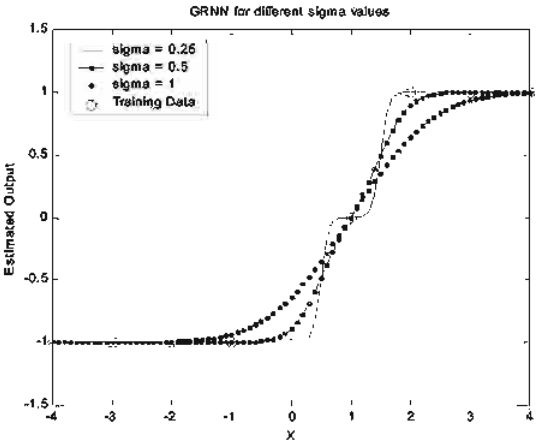


Figure 10: Example of GRNN application



difficulty to analyze and to provide a clear understanding of how its behavior is related to its contents. The inexistence of an intuitive method for selecting the optimal smoothing parameter is also a difficult task to solve.

GRNN-Based Controller

The non-linear control helicopter and robotic systems are particularly good application areas that can be used to demonstrate the potential speed of neural networks implemented in parallel hardware and the adaptability of instant learning. First, the GRNN learns the relation between the state vector of the system and the control variables. After the GRNN-based model is trained, it can be used to determine control inputs. One way in which the neural network could be used to control a system is shown in Figure 11.

The GRNN is not trained in the traditional neural network sense where weights are adjusted iteratively. Rather, all training data is stored in memory (thus the term memory-based network) and only when the output is necessary for a different input

Figure 11: A GRNN controller

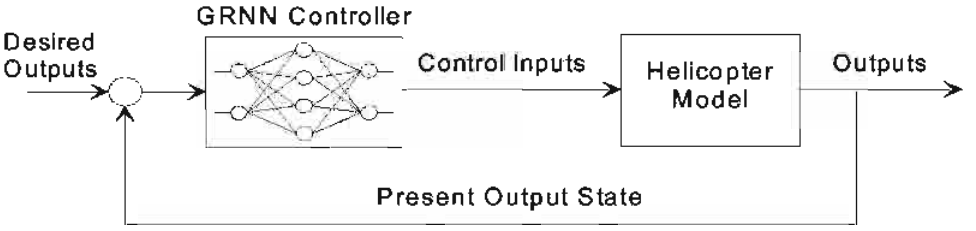


Table 1: Vector prototype for each control input

Control Inputs	State Variables
Collective	$\{\delta_a, z, w\}$
Longitudinal cyclic	$\{\delta_b, x, u, \theta, q\}$
Lateral cyclic	$\{\delta_c, y, v, \phi, p\}$
Pedals	$\{\delta_d, \varphi, r\}$

a new computation is performed. In controlling the helicopter each data training is a vector with the input variables and the corresponding output for each controller. Only the samples that represent the cluster centers are used to populate the network. The reduction of training data used by the GRNN is an important problem because we can obtain a faster controller maintaining a good performance. It is also possible to separate the obtained representative clusters' center, for example, in two data sets for the coarse and fine control.

To control the helicopter flight mode in hover position, four data sets corresponding to each input control were used. In each data set exists a set of vectors that correspond to the representative clusters obtained after the clustering algorithm is applied. The vector structure in each data set is given in Table 1.

SIMULATED RESULTS

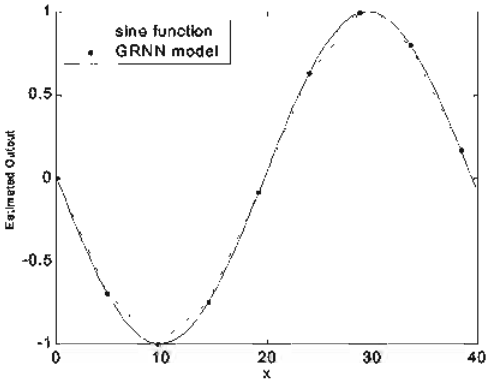
An experiment was performed to determine the extent to which performance of the clustering algorithm degrades with less training data. Figure 12 shows the output of the sine function and the model when the identification procedure was carried out for only nine patterns instead of the fifty-five used to represent the sine function.

Figures 13 and 14 show the open loop responses of the helicopter displacements and euler angles corresponding to impulse control inputs in the longitudinal and lateral cyclic. First it was applied an impulse in the longitudinal control and then in the lateral control input. The initial conditions for the helicopter model are as follows:

$$u_0 = 0,\ v_0 = 0, w_0 = 0,\ \phi_0 = -0.0576,\ \theta_0 = 0.0126,\ \varphi_0 = 0.$$

The initial conditions corresponding to the derivatives of the state variables described above are zero.

Figure 12: Output of sine function (solid line) and of GRNN model (dotted line) after training with only nine patterns



Figures 15 to 23 shows the system response using the GRNN controller using the data set for each controller. Each data set contains the representative clusters obtained after the clustering process. Figures 15 to 17 show the displacement of the helicopter in the longitudinal, lateral and vertical axis, respectively. These three Figures show that the higher displacement changes occur in the forward and lateral axis rather than in the vertical axis. This happens because the impulse control inputs were applied to the longitudinal and lateral cyclic which are the commands to control the forward and lateral displacements of the helicopter.

Figure 18 shows the trajectory of the helicopter in the 2D plan. The arrows indicate the direction of the helicopter displacement after applied impulses in the control inputs. After approximately five minutes the helicopter is stabilized in the initial position (i.e. $x = y = z = 0$).

Figure 13: Simulated results of (x, y, z) to an impulse control input, in the longitudinal cyclic

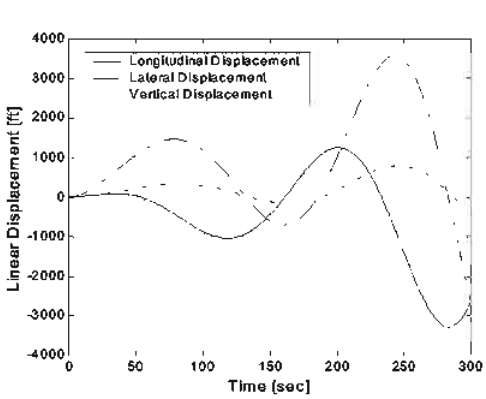


Figure 14: Simulated results of (ϕ, θ, φ) to an impulse control input, in the longitudinal cyclic

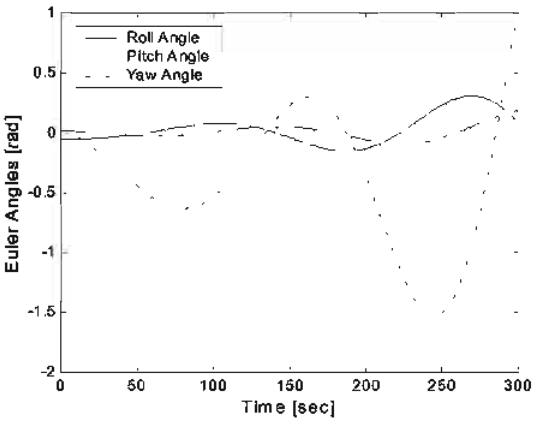


Figure 19 shows the roll, pitch and yaw angles. Even when the initial conditions of the roll and pitch angles are different from zero, these angles stabilize, permitting the control of the helicopter.

Figures 20 to 23 show the control inputs applied to the helicopter. The control inputs were limited to ±5V for simulate practical limitations of the actuators. Since the higher perturbation occurs in the longitudinal and lateral displacements than it was the longitudinal and lateral cyclic control inputs the actuators with higher performance.

Figure 15: Simulated results of x for an impulse control input, in the longitudinal and lateral cyclic

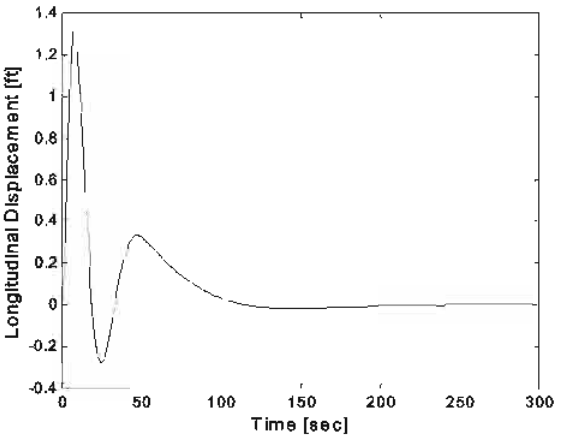


Figure 16: Simulated results of y for impulse control inputs, in the longitudinal and lateral cyclic

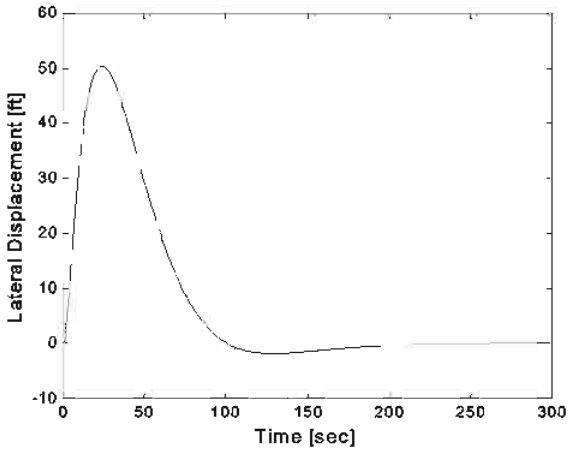
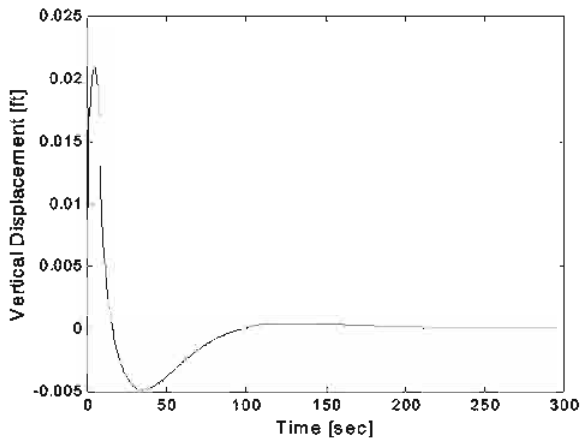


Figure 17: Simulated results of z for impulse control inputs, in the longitudinal and lateral cyclic



Since for each flight mode it can be used one distinct GRNN controller, then it is not necessary to reduce the number of clusters. Each controller has a cluster set that represents the dynamic behavior of the helicopter for the specific flight mode.

CONCLUSIONS

To control the displacement of a single main rotor helicopter of a 15,000-pound using the longitudinal, lateral and collective control inputs, three GRNN

Figure 18: Trajectory of the helicopter in the (x, y) plan

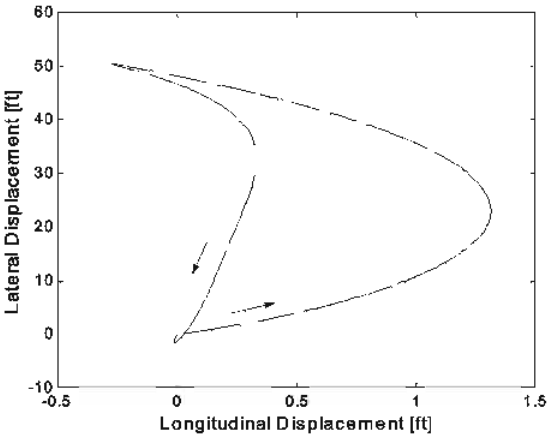
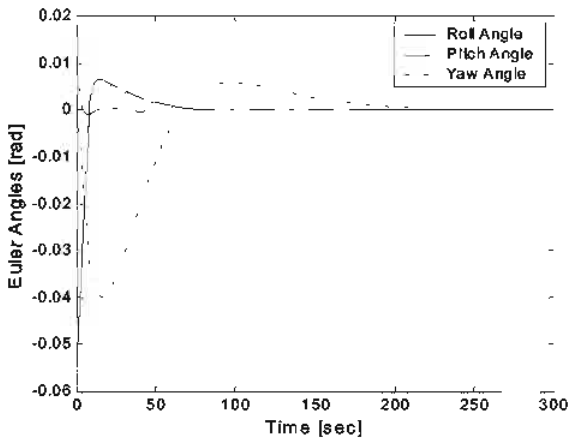


Figure 19: Simulated results of (ϕ, θ, φ) to an impulse control input, in the longitudinal and lateral cyclic



controllers have been used. The direction of the helicopter nose was controlled by the pedals control input using another GRNN controller. With these controllers it was possible to enable the helicopter to maintain its stationary position for a long period of time. The advantage of the GRNN controller is the fast-learning capability and the non iterative process. For many gathered training instances, the computation amount became large. Therefore, to overcome this problem a clustering algorithm was implemented.

Figure 20: Simulated result of the collective control input

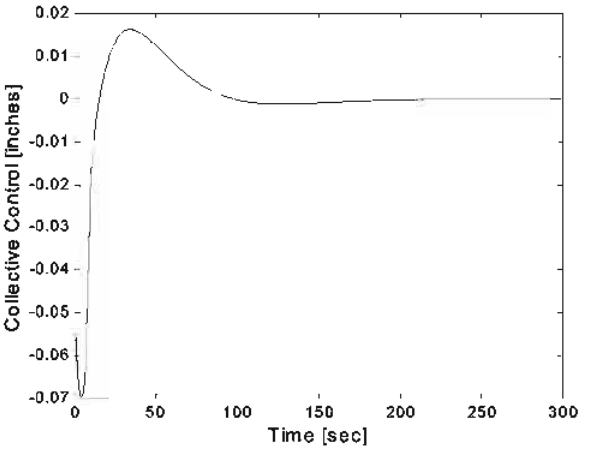
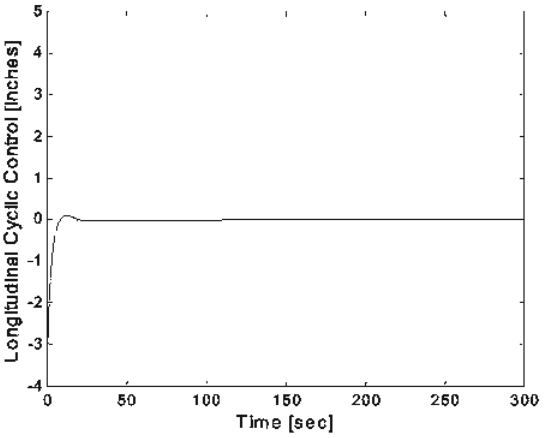


Figure 21: Simulated result of the longitudinal cyclic control



FUTURE DIRECTIONS

A fuzzy algorithm can also control the helicopter. Fuzzy rule base systems are linguistic in nature and can be inspected by a human expert. However, GRNN and fuzzy algorithms could be used together. Fuzzy logic gives a common framework for combining numerical training data and expert linguistic knowledge along with the compact transparency and computational efficiency of rule bases, while the GRNN

Figure 22: Simulated result of the lateral cyclic control

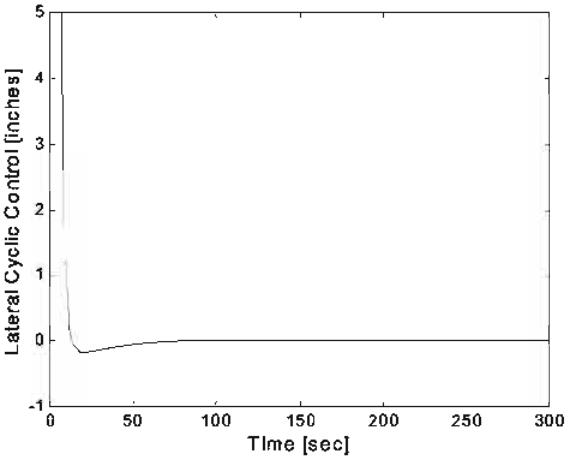
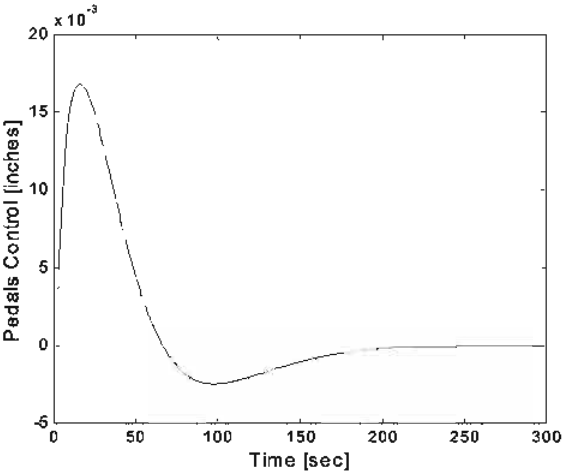


Figure 23: Simulated result of the pedals control



gives the approach rapid adaptive capability. For this reason, one of the future research directions may be the hybrid fuzzy logic/GRNN approach.

ACKNOWLEDGMENTS

The authors would like to thank Professor Mark Dreier from Bell Helicopter Textron, USA. He graciously sent to us his Matlab/Simulink system containing a helicopter model describing the dynamic behaviour of the helicopter in hover flight mode.

REFERENCES

Burrascano, P. (1991, July). Learning vector quantization for the probabilistic neural network, *IEEE Trans. Neural Network,* (2), 458-461.

Chen, C. H. (1996). *Fuzzy Logic and Neural Network Handbook.* McGraw-Hill.

Fagg, A., Lewis, M., Montgomery, J. & Bekey, G. (1993). The USC autonomous flying vehicle: An experiment in real-time behavior-based control. *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems,* Yokohama, Japan, July.

Larkin, L. (1984). A fuzzy logic controller for aircraft flight control. *Proceedings 23rd Conference on Decision and Control.* Las Vegas, NV. December.

Montgomery, J. F. (1999). Learning Helicopter Control through Teaching by Showing. PhD dissertation. University of Southern California, Los Angeles, CA.

Moody, J., & Darken, C. (1989). Fast learning in networks of locally-tuned processing units, *Neural Computation,* (1). 281-294.

Nadaraya, E. A. (1964), On estimating regression, *Theory Probab. Application* 10, 186-190.

Phillips, C., Karr, C. & Walker, G. (1994). A genetic algorithm for discovering fuzzy logic rules. *Proceedings International Fuzzy Systems and Intelligent Controls Conference,* March.

Specht, D.F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks,* 2(6), November, 568-576.

Sugeno, M. (1997). *The Industrial Electronics Handbook.* CRC Press, 1127-1138.

Sugeno, M. (1998). Recent advances in fuzzy control: Stability issues and application to an unmanned helicopter. *World Automation Congress,* Alaska, May.

Tou, J. T. & Gonzalez, R. C. (1974). *Pattern Recognition Principles.* Reading, MA: Addison-Wesley.

Tsoukalas, L.H. & Uhrig, R.E. (1997). *Fuzzy and Neural Approaches in Engineering.* A volume in the Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications, and Control. Simon Haykin, Series Editor.

Wade, R. & Walker, G. (1994). Fuzzy logic adaptive controller-helicopter (FLAC-H): A multi-platform, rotary-winged aerial robotic control system. *19th Army Science Conference.* Orlando, FL, June.

Wade, R., Walker, G. & Phillips, C. (1994). Combining genetic algorithms and

aircraft simulations to tune fuzzy rules in a helicopter control system. *Advances in Modelling and Simulation Conference,* Huntsville, AL, April.

Walker, G. & Mo, S. (1994). Forward modelling of helicopter flight dynamics using recurrent neural networks, *19th Army Science Conference*, Orlando, FL, June.

Chapter IV

A Biologically Inspired Neural Network Approach to Real-Time Map Building and Path Planning

Simon X. Yang
University of Guelph, Canada

ABSTRACT

A novel biologically inspired neural network approach is proposed for real-time simultaneous map building and path planning with limited sensor information in a non-stationary environment. The dynamics of each neuron is characterized by a shunting equation with both excitatory and inhibitory connections. There are only local connections in the proposed neural network. The map of the environment is built during the real-time robot navigation with its sensor information that is limited to a short range. The real-time robot path is generated through the dynamic activity landscape of the neural network. The effectiveness and the efficiency are demonstrated by simulation studies.

INTRODUCTION

Real-time path planning with collision free in a non-stationary environment is a very important issue in robotics. There are a lot of studies on the path planning for robots using various approaches. Most of the previous models use global methods to search the possible paths in the workspace (e.g., Lozano-Perez, 1983; Zelinsky,

1994; Al-Sultan & Aliyu, 1996; Li & Bui, 1998). Ong and Gilbert (1998) proposed a new searching-based model for path planning with penetration growth distance, which searches over collision paths instead of the free workspace. Most searching-based models can deal with static environment only and are computationally complicated when the environment is complex. Some of the early models deal with static environment only, and may suffer from undesired local minima (e.g., Ilari & Torras, 1990; Zelinsky, 1994; Glasius et al., 1994). Some previous robot motion planning models require the prior information of the non-stationary environment, including the varying target and obstacles. For example, Chang and Song (1997) proposed a virtual force guidance model for dynamic motion planning of a mobile robot in a predictable environment, where an artificial neural network is used to predict the future environment through a relative-error-back-propagation learning.

Several neural network models were proposed to generate real-time trajectory through learning (e.g., Li & Ogmen, 1994; Beom & Cho, 1995; Glasius et al., 1994; 1995; Zalama, Gaudiano & Lopez Coronado, 1995; Chang & Song, 1997; Gaudiano et al., 1996; Yang, 1999; Yang & Meng, 2000a, 2000b, 2001). The learning based approaches suffer from extra computational cost because of the learning procedures. In addition, the planned robot motion using learning based approaches is not optimal, especially during the initial learning phase of the neural network. For example, Zalama et al. (1995) proposed a neural network model for the navigation of a mobile robot, which can generate dynamical trajectory with obstacle avoidance through unsupervised learning.

Glasius et al. (1995) proposed a neural network model for real-time trajectory formation with collision free in a non-stationary environment. However, this model suffers from slow dynamics and cannot perform properly in a fast changing environment. Inspired by Hodgkin and Huxley's (1952) membrane equation and the later developed Grossberg's (1988) shunting model, Yang and Meng (2000a) proposed a neural network approach to dynamical trajectory generation with collision free in an arbitrarily changing environment. These models are capable of planning a real-time optimal path in non-stationary situations without any learning process. But the planned paths in Glasius et al. (1995) and Yang and Meng (2000a) do not take into account the clearance from obstacles, which is demanded in many situations. By introducing inhibitory lateral connections in the neural network, Yang and Meng (2000b) proposed a new model for path planning with safety consideration, which is capable of generating a "comfortable" path for a mobile robot, without suffering either the "too close" (narrow safety margin) or the "too far" (waste) problems. However, the models in Ilari and Torras (1990), Zelinsky (1994), Zalama et al. (1995), Glasius et al. (1995) and Yang and Meng (2000a, 2000b) assume that the workspace is known, which is not practically feasible in many applications.

In this chapter, a novel biologically inspired neural network approach, based on the model in Yang and Meng (2000b) for path planning of mobile robots with completely known environment, is proposed for real-time simultaneous map building and path planning of mobile robots in a dynamic environment, where the environment is assumed completely unknown. The state space of the topologically organized neural network is the Cartesian workspace, where the dynamics of each neuron is characterized by a shunting equation that was derived from Hodgkin and Huxley's (1952) membrane model for a biological system. The robot navigation is based on the target location, and robot sensor readings that are limited to a short range. The real-time robot path is generated from the neural activity landscape of the neural network that adapts changes according to the target location and the known map of the workspace. A map of the environment is built in real time when the robot is moving toward the target, where the sensor readings are obtained from the onboard sensors of the mobile robot that are limited to a certain local range only.

THE MODEL

In this section, the originality of the proposed neural network approach is briefly introduced. Then, the philosophy of the proposed neural network approach and the model algorithm are presented. Finally, the stability of the proposed model is proven using both qualitative analysis and a Lyapunov stability theory.

Originality

Hodgkin and Huxley (1952) proposed a computational model for a patch of membrane in a biological neural system using electrical circuit elements. This modeling work, together with other experimental work, led them to a Nobel Prize in 1963 for their discoveries concerning the ionic mechanisms involved in excitation and inhibition in the peripheral and central portions of the nerve cell membrane. In Hodgkin and Huxley's (1952) membrane model, the dynamics of voltage across the membrane, V_m, is described using a state equation technique such as:

$$C_m \frac{dV_m}{dt} = -(E_p + V_m)g_p + (E_{Na} + V_m)g_{Na} - (E_K + V_m)g_K \qquad (1)$$

where C_m is the membrane capacitance. Parameters E_K, E_{Na} and E_p are the Nernst potentials (saturation potentials) for potassium ions, sodium ions and the passive leak current in the membrane, respectively. Parameters g_K, g_{Na} and g_p represent the conductance of potassium, sodium and passive channels, respectively. This model

provided the foundation of the shunting model and led to a lot of model variations and applications (Grossberg, 1988).

By setting $C_m = 1$, substituting $x_i = E_p + V_m$, $A = g_p$, $B = E_{Na} + E_p$, $D = E_k - E_p$, $S_i^e = g_{Na}$ and $S_i^i = g_K$ in Eqn. (1), a typical shunting equation is obtained (Ogmen & Gagne, 1990a, 1990b) as:

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)S_i^e(t) - (D + x_i)S_i^i(t) \qquad (2)$$

where variable x_i is the neural activity (membrane potential) of the i-th neuron. Parameters A, B and D are non-negative constants representing the passive decay rate, the upper and lower bounds of the neural activity, respectively. Variables S_i^e and S_i^i are the excitatory and inhibitory inputs to the neuron (Ogmen & Gagne, 1990a, 1990b; Yang, 1999). This shunting model was first proposed by Grossberg to understand the real-time adaptive behavior of individuals to complex and dynamic environmental contingencies (Grossberg, 1973, 1982, 1983, 1988), and has a lot of applications in biological and machine vision, sensory motor control, and many other areas (e.g., Grossberg, 1982, 1988; Ogmen & Gagne, 1990a, 1990b; Ogmen, 1993; Zalama et al., 1995; Gaudiano et al., 1996; Yang, 1999).

Model Algorithm

The fundamental concept of the proposed model is to develop a neural network architecture, whose dynamic neural activity landscape represents the limited knowledge of the dynamically varying environment from onboard robot sensors. By properly defining the external inputs from the varying environment and internal neural connections, the target and obstacles are guaranteed to stay at the peak and the valley of the activity landscape of the neural network, respectively. The target globally attracts the robot in the whole state space through neural activity propagation, while the obstacles have only local effect in a small region to avoid collisions and to achieve the clearance from obstacles. The real-time collision-free robot motion is planned through the dynamic activity landscape of the neural network.

The neural network architecture of the proposed model is a discrete topologically organized map that is used in several neural network models (Glasius et al., 1995; Yang & Meng, 2000a, 2000b). The proposed model is expressed in a finite (F-) dimensional (F-D) state space, which can be either the Cartesian workspace or the configuration joint space of a multi-joint manipulator. The location of the i-th neuron at the grid in the F-D state space, denoted by a vector $q_i \in R^F$, represents

a position in the workspace or a configuration in the joint space. The target globally attracts the robot through neural activity propagation, while the obstacles push the robot only locally in a small region to avoid collision. To take into account the clearance from obstacles, there are both excitatory and inhibitory lateral connections. The dynamics of the i-th neuron in the neuron network is given by a shunting equation:

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)\left([I_i]^+ + \sum_{j=1}^{k} w_{ij}[x_j]^+ \right) - (D + x_i)\left([I_i]^- + \sum_{j=1}^{k} v_{ij}[x_j - \sigma]^- \right) \quad (3)$$

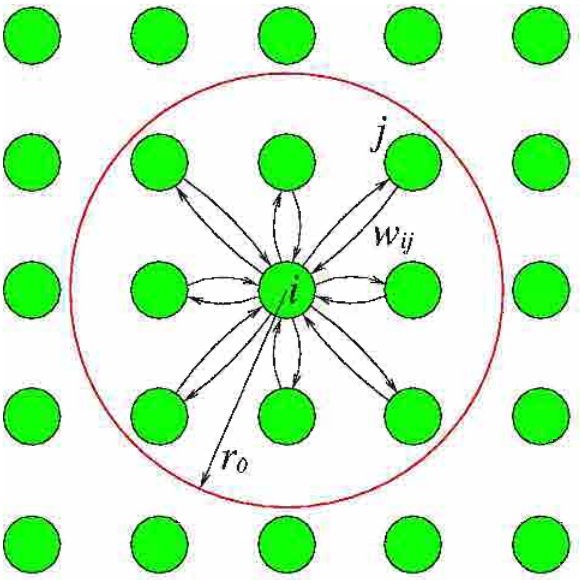
where the excitatory and inhibitory inputs are $[I]^+ + \sum_{i=1}^{k} w_{ij}[x_j]^+$ and $[I]^- + \sum_{i=1}^{k} w_{ij}[x_j]^-$, respectively. The external input I_i to the i-th neuron is defined as: $I_i = E$, if there is a target; $I_i = -E$, if there is an obstacle; $I_i = 0$, otherwise, where E is a very large positive constant over its total lateral input. Unlike those models in Yang and Meng (2000a, 2000b) where the whole environment is assumed to be completely known, the proposed model assumes that initially the environment is *completely unknown*, except that the robot knows the target location. Thus the external input I_i depends on the known information of the environment from its onboard sensors whose capacity is limited to a certain local range. A map of the environment is building from the sensor information during the real-time robot navigation.

The function $[a]^+$ is a linear-above-threshold function defined as, $[a]^+ = \max(a,0)$, and the non-linear function $[a]^-$ is defined as $[a]^- = \max(-a,0)$. The weights of the excitatory and inhibitory connections, w_{ij} and v_{ij}, from the i-th neuron to the j-th neuron are defined as:

$$w_{ij} = f(\| q_i - q_j \|) \text{ and } v_{ij} = \beta w_{ij} \quad (4)$$

respectively, where β is a positive constant, $\beta \in [0,1]$, and $\|q_i - q_j\|$ represents the Euclidean distance between vectors q_i and q_j in the state space. Function $f(a)$ is a monotonically decreasing function, such as a function defined as: $f(a) = \mu/a$, if $0 < a < r_0$; $f(a) = 0$, if $a \geq r_0$, where μ and r_0 are positive constants. Therefore, it is obvious that the neural connection weights w_{ij} and v_{ij} are symmetric. The neuron has only local connections in a small region $(0,r_0)$, i.e., its receptive field is the space whose distance to the i-th neuron is less than r_0. The neurons located within the receptive field of the i-th neuron are referred as its *neighboring neurons*. The parameter k is the total number of neighboring neurons of the i-th neuron. Parameter σ is the threshold of the inhibitory lateral neural connections. The threshold of the

Figure 1: Schematic diagram of the neural network for robot path planning when the state space is 2D; the i-th neuron has only 8 lateral connections to its neighboring neurons that are within its receptive field



excitatory connections is chosen as a constant zero. A schematic diagram of the neural network in 2D is shown in Figure 1, where r_0 is chosen as $r_0=2$. The receptive field of the i-th neuron is represented by a circle with a radius of r_0.

The proposed neural network characterized by Eqn. (3) guarantees that the positive neural activity can propagate to the whole state space. However, the negative activity stays locally only in a small region, due to the existence of the threshold σ of the inhibitory lateral connections. Therefore, the target *globally* influences the whole state space to attract the robot, while the obstacles have only *local* effect to avoid collision. In addition, by choosing different β and/or σ values, the local influence from the obstacles is adjusted, and a suitable strength of clearance from obstacles is selected. Therefore, the proposed model is capable of planning the shortest path from the starting position to the target, or a safer path, or the safest path, depending on the different requirement.

The positions of the target and obstacles may vary with time. The activity landscape of the neuron network dynamically changes due to the varying external inputs and the internal lateral connections. The optimal path is generated from the dynamic activity landscape by a gradient ascent rule. For a given *present position* in the workspace or in the robot manipulator joint space, denoted by q_p, the *next position* q_n (also called "command position") is obtained by:

$$p_n \Leftarrow x_{p_n} = \max\{x_j, j = 1,2,...,k\} \tag{5}$$

where k is the number of the neighboring neurons, i.e., all the possible next positions of the present position q_p. After the *present position* reaches its *next position*, the *next position* becomes a *new present position*. The present position *adaptively* changes according to the varying environment. The speed of the robot can be defined as a function of its distance to the nearest obstacle, e.g., a function defined as:

$$v = \begin{cases} v_m, & \text{if } d \geq d_0 \\ v_m d / d_0, & \text{otherwise} \end{cases} \tag{6}$$

where v_m is the maximum robot speed, d_0 is a positive constant and d_0 is the Euclidean distance from robot to its nearest obstacle.

The dynamic activity landscape of the topologically organized neural network is used to determine *where* the next robot position should be. However, *when* the robot moves to the next position is determined by the robot moving speed. In a static environment, the activity landscape of the neural network will reach a steady state, which will later be proven using the Lyapunov stability theory. Mostly the robot reaches the target much earlier than the activity landscape reaches the steady state of the neural network. When a robot is in a dynamically changing environment, the neural activity landscape will never reach a steady state. Due to the very large external input constant E, the target and the obstacles stay at the peak and the valley of the activity landscape of the neural network, respectively. The robot keeps moving toward the target with obstacle avoidance till the designated objectives are achieved.

Stability Analysis

In the shunting model in Eqn. (2) and (3), the neural activity x_i increases at a rate of $(B-x_i)S_i^e$, which is not only proportional to the excitatory input S_i^e, but also proportional to an auto gain control term $(B-x_i)$. Thus, with an equal amount of input S_i^e, the closer the values of x_i and B are, the slower x_i increases. When the activity x_i is below B, the excitatory term is positive, causing an increase in the neural activity. If x_i is equal to B, the excitatory term becomes zero, and x_i will no longer increase no matter how strong the excitatory input is. In case the activity x_i exceeds B, $B - x_i$ becomes negative and the shunting term pulls x_i back to B. Therefore, x_i is forced to stay below B, the upper bound of the neural activity. Similarly, the inhibitory term forces the neural activity to stay above the lower bound $-D$.

Therefore, once the activity goes into the finite region $[-D, B]$, it is guaranteed that the neural activity will stay in this region for any value of the total excitatory and inhibitory inputs (Yang, 1999).

The stability and convergence of the proposed model can also be rigorously proven using a Lyapunov stability theory. From the definition of $[a]^+$, $[a]^-$ and v_{ij}, Eqn. (3) is rewritten into Grossberg's general form (Grossberg, 1988):

$$\frac{dx_i}{dt} = a(x_i)\left(a(x_i) - \sum_{j=1}^{N} c_{ij} d(x_i) \right) \tag{7}$$

by the following substitutions:

$$a_i(x_i) = \begin{cases} B - x_i, & \text{if } x_j \geq 0 \\ D + x_i & \text{if } x_j < 0 \end{cases}$$

$$b_i(x_i) = \frac{1}{a_i(x_i)} \left(B[I_i]^+ - C[I_i]^- - (A + [I_i]^+ + [I_i]^-) x_i \right)$$

$$c_{ij} = -w_{ij}$$

and

$$d_j(x_j) = \begin{cases} x_j, & \text{if } x_j \geq 0 \\ \beta(x_j - \sigma), & \text{if } x_j < \sigma \\ 0, & \text{otherwise} \end{cases}$$

Since the neural connection weight is symmetric, $w_{ij} = w_{ji}$, then $c_{ij} = c_{ji}$ (symmetry). Since B and D are non-negative constants and $x_i \in [-D, B]$, then ai(xi) then $a_i(x_i) \geq 0$ (positivity). Since $d'_j(x_j) = 1$ at $x_j > 0$; $d'_j(x_j) = \beta$ at $x_j > \sigma$; and $d'_j(x_j) = 0$, otherwise, then $d_j(x_j) \geq 0$ (monotonicity). Therefore, Eqn. (5) satisfies all the three stability conditions required by Grossberg's general form (Grossberg, 1988). The Lyapunov function candidate for Eqn. (7) can be chosen as:

$$v = -\sum_{i=1}^{N} \int^{x_i} b_i(y_i) d_i'(y_i) dy_i + \frac{1}{2} \sum_{j,k=1}^{N} c_{kj} d_j(x_i) d_k(x_k) \tag{8}$$

The derivative of v along all the trajectories is given as:

$$\frac{dv}{dt} = -\sum_{j=1}^{N} a_i d_i' (b_i - \sum_{j=1}^{N} c_{ij} d_j)^2$$

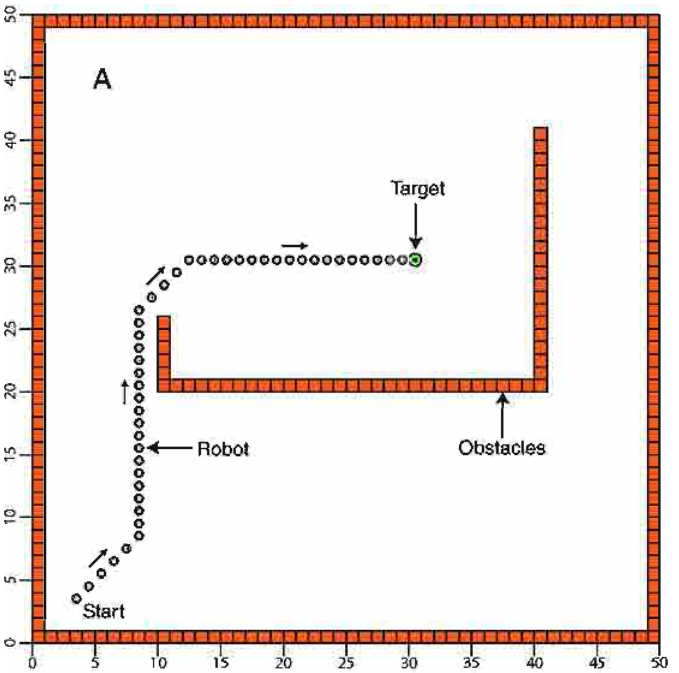
Since $a_i \geq 0$ and $d_i' \geq 0$, then $dv/dt \leq 0$ along all the trajectories. The rigorous proof of the stability and convergence of Eqn. (7) can be found in Grossberg (1983). Therefore, the proposed neural network system is stable. The dynamics of the network is guaranteed to converge to an equilibrium state of the system.

SIMULATIONS

To demonstrate the effectiveness of the proposed neural network model, the proposed neural network model for real-time map building and path planning is applied to a room-like environment in Figure 2A, where the static obstacles are shown by solid squares. The target is located at position (30,30), while the starting position of the robot is at (4,4). The neural network has 50×50 topologically organized neurons, with zero initial neural activities. The model parameters are chosen as: $A = 10$ and $B = D = 1$ for the shunting equation; μ=1, β=0.8, σ = –0.8 and r_0=2 for the lateral connections; and E=100 for the external inputs. Three cases are carried out. In the first case, same as the models in Yang and Meng (2000a, 2000b), it is assumed that the environment is completely known by some additional sensors in the workspace. The generated robot path is shown in Figure 2A, where the robot is represented by circles. It shows that the robot can reach the target without obstacle avoidance. The neural activity landscape when the robot arrives at the target location is shown in Figure 2A, where the peak is at the target location, while the valleys are at the obstacle locations.

In the second case, the environment is assumed to be completely unknown, except the target location. The onboard robot sensors can "see" in a limited range within a radius of R=5 (see the circle in lower right corner of Figure 3A). As shown in Figure 3A, initially the robot sees some obstacles on its back, but there are no obstacles in its front direction toward the target. Thus the robot moves straight forward to the target. However, when the robot arrives at location (16,16), it starts to sense the obstacle in its front. When the robot arrives at (18,18), the built map is shown in Figure 3A, where a few obstacles are detected by its onboard sensors, and the activity landscape is shown in Figure 3B. The robot is moving toward the target along a collision-free path; more and more obstacles were detected. When the robot arrives at (18,41), the built map and the activity landscape are shown in

Figure 2: Path planning with safety consideration when the environment is completely known--A: the dynamic robot path; B: the activity landscape when the robot reaches the target



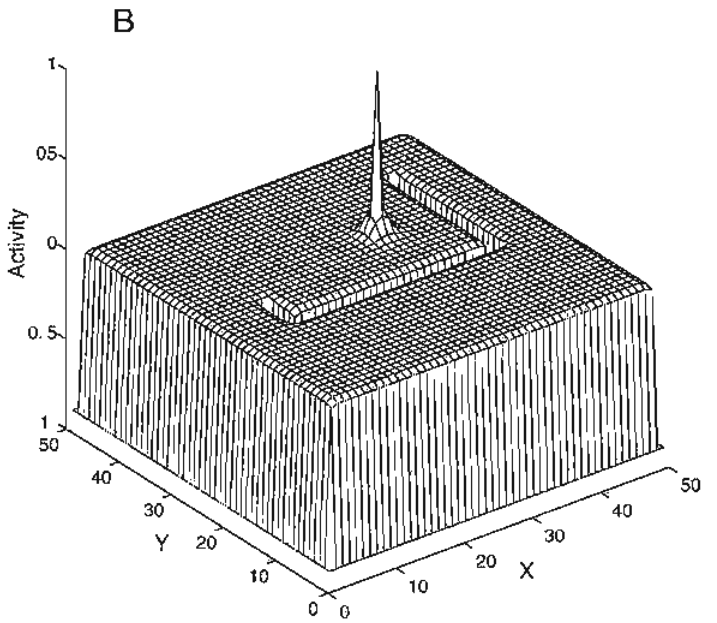
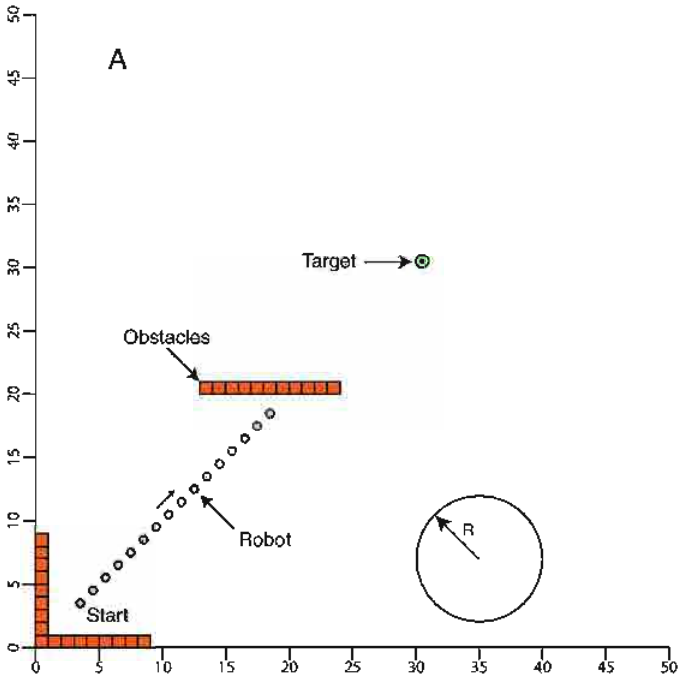


Figure 3: Map building and path planning with sensor information limited to a radius or R=5--A and B: the dynamic robot path and the environment map (A) and the neural activity landscape (B) when the robot arrives at (18,18); C and D: the dynamic robot path and the environment map (C) and the neural activity landscape (D) when the robot arrives at (18,41)



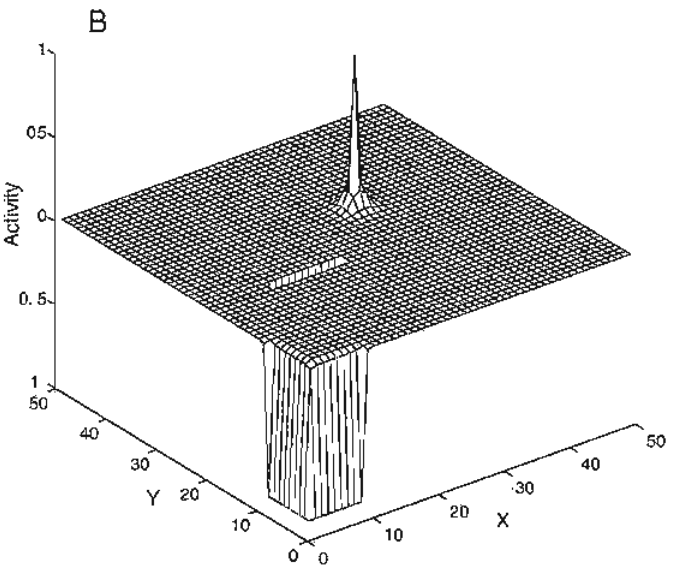
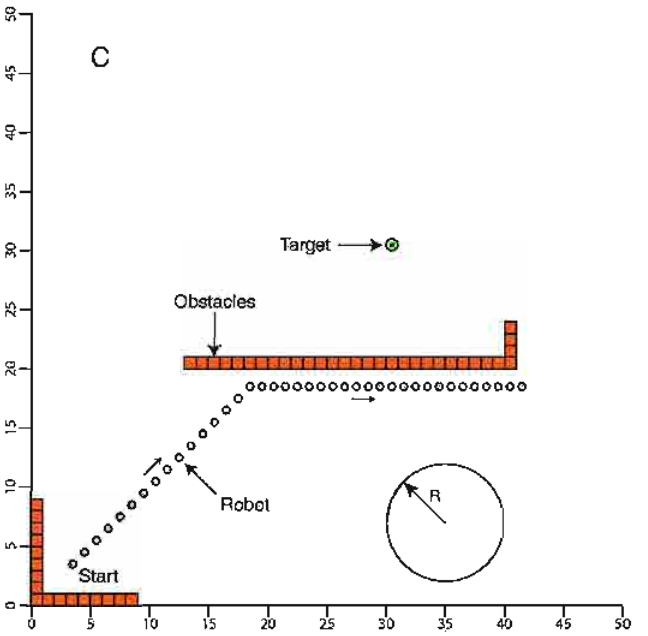


Figure 3: (continued) (C) the neural activity landscape (D) when the robot arrives at (18,41)



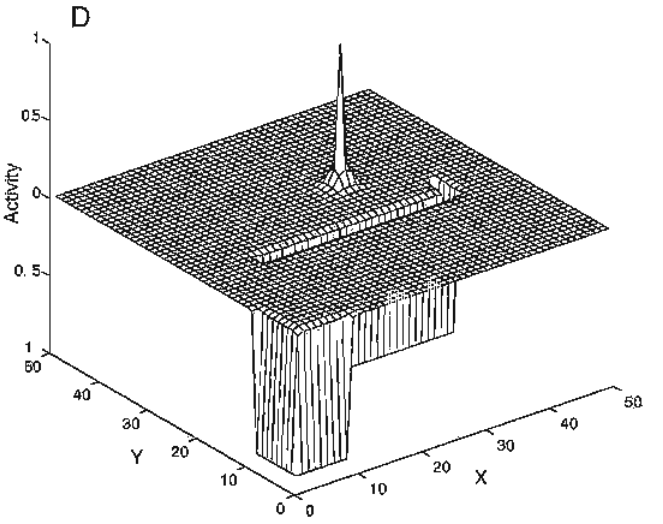
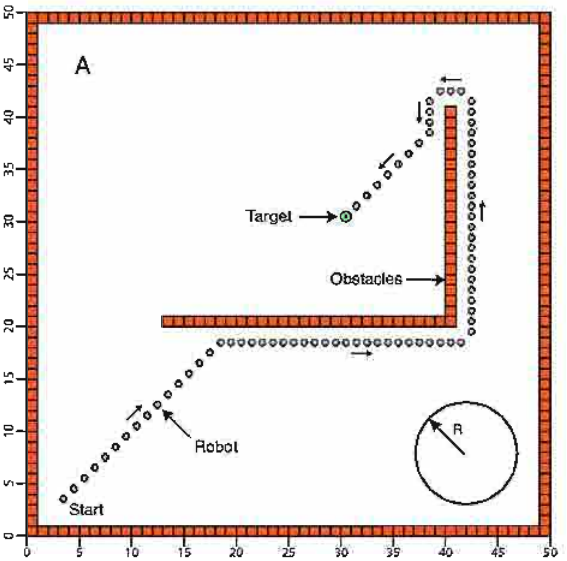


Figure 4: Map building and path planning with limited sensor information-- A: the robot path and the built map in same case in Figure 3; B: the robot path and the built map when there are obstacles suddenly placed in its front to close to the gate to the target when the robot arrives at (42,41) (marked by an arrow) in the case in left panel



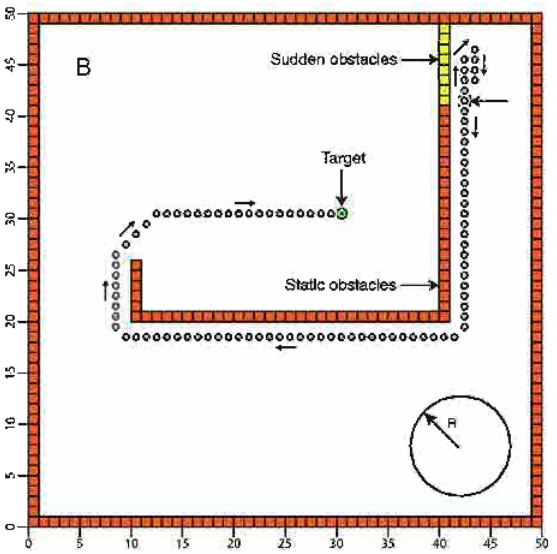


Figure 3C and Figure 3D, respectively. The robot continues to move toward the target. The robot traveling path to reach the target and the built map are shown in Figure 3A.

In the third case, the condition initially is the same as in Case 2. However, when the robot arrives at (42, 41), there are obstacles suddenly placed in front of the robot, which close the gate for the robot to reach the target. The robot has to move around there, and finally the robot has to move back, pass around the obstacles and finally reach the target from the other side. The robot path after the sudden obstacles were placed is shown in Figure 4B. It shows that the robot is capable of reaching the target with obstacle clearance.

DISCUSSIONS

In this section, the parameter sensitivity of the proposed neural network model will be discussed. Then a simple model characterized by an additive equation is obtained from the proposed shunting model by removing the auto gain control terms and lumping together the excitatory and inhibitory inputs.

Parameter Sensitivity

Parameter sensitivity is a very important factor to be considered when a model is proposed or evaluated. An acceptable model should be robust, i.e., not very sensitive to changes in its parameter values. There are only few parameters in the proposed model in Eqn. (3). The upper and lower activity bounds B and D, the receptive field parameter r_0 and the external input constant E are not important factors. The passive decay rate A determines the transient response of the neurons, which is very important for the model dynamics, particularly when the target and the obstacle are varying fast. The lateral connection weight parameter μ is also an important factor, which determines the propagation of the neural activity in the neural network. The relative inhibitory lateral connection parameter β and the threshold of the inhibitory connections σ determine the strength of the clearance from obstacles. They are very important factors as well. A detailed discussion through description and simulation of the model parameters can be found in Yang and Meng (2000b, 2001).

The proposed model is not very sensitive to the variations of model parameters and the connection weight function. The parameters can be chosen in a very wide range. The weight function can be any monotonically decreasing function (Yang & Meng, 2001).

Model Variation

If the excitatory and inhibitory connections in the shunting equation in Eqn. (3) are lumped together and the auto gain control terms are removed, then a simpler form can be obtained from Eqn. (3):

$$\frac{dx_i}{dt} = -Ax_i + I_i + \sum_{j=1}^{k} w_{ij}[x_j]^+ - \sum_{j=1}^{k} v_{ij}[x_j - \sigma]^-, \tag{9}$$

This is an additive equation (Grossberg, 1988). The term $I_i + \sum_{j=1}^{k} w_{ij}[x_j]^+ - \sum_{j=1}^{k} v_{ij}[x_j - \sigma]^-$ represents the total inputs to the i-th neuron from the external and internal connections. The non-linear functions $[a]^+$, $[a]^-$ and the threshold σ are defined as the same as earlier in this chapter, which together guarantee that the positive neural activity can propagate to the whole workspace while the negative activity can propagate locally in a small region only. Thus the target globally attracts the robot in the whole workspace, while the obstacles have only local effects to achieve clearance from obstacles. Therefore this additive model satisfies the fundamental concepts of the proposed approach described earlier in this chapter. It is capable of simultaneously planning robot path and building environment map in most situations. From the definition of $[a]^+$, $[a]^-$ and v_{ij} Eqn. (9) can be further rewritten into a compact form as:

$$\frac{dx_i}{dt} = -Ax_i + I_i + \sum_{j=1}^{k} w_{ij} d(x_j) \tag{10}$$

where $d(x_j)$ is defined in Eqn. (7). The stability of this additive model can also be proven using a Lyapunov stability theory, although its neural activity does not have any bounds. Eqn. (9) can be rewritten into Grossberg's general form in Eqn. (7) by variable substitutions. It is easy to prove that Eqn. (9) satisfies all the three stability conditions of Eqn. (7) (Grossberg, 1988; Yang, 1999, Yang & Meng, 2001). Therefore this additive neural network system is stable.

There are many important differences between the shunting model and the additive model, although the additive model is computationally simpler. By rewriting them into the general form in Eqn. (7), unlike the additive model in Eqn. (9), the amplification function $a_i(x_i)$ of the shunting model in Eqn. (3) is not a constant, and the self-signal function $b_i(x_i)$ is non-linear. The shunting model in Eqn. (3) has two *auto gain control* terms, $(B-x_i)$ and $(D+x_i)$, which result in that the dynamics of

Eqn. (3) remain sensitive to input fluctuations (Grossberg, 1988). Such a property is important for the real-time robot path planning when the target and obstacles are varying. In contrast, the dynamics of the additive equation may saturate in many situations (Grossberg, 1988). Furthermore, the activity of the shunting model is bounded in the finite interval $[-D, B]$, while the activity in the additive model does not have any bounds (Grossberg, 1988; Ogmen & Gagne, 1990a, 1990b; Yang & Meng, 2000a, 2000b, 2001).

CONCLUSION

In this chapter, a novel biologically inspired neural network model is proposed for the real-time map building and path planning with safety consideration. Several points are worth noticing about the proposed model:

- The strength of the clearance from obstacles is adjustable. By changing suitable model parameters, this model is capable of planning the shortest path, or a comfortable path, or the safest path (Yang & Meng, 2000b).
- The algorithm is computationally efficient. The map is built during the robot navigation, and the robot path is planned through the dynamic neural activity landscape *without* any prior knowledge of the dynamic environment, *without* explicitly searching over the free space or the collision paths, *without* explicitly optimizing any cost functions and *without* any learning procedures.
- The model can perform properly in an arbitrarily varying environment, even with a sudden environmental change, such as suddenly adding or removing obstacles.
- The model is biologically plausible. The neural activity is a continuous analog signal and has both upper and lower bounds. In addition, the continuous activity prevents the possible oscillations related to parallel dynamics of discrete neurons (Glasius et al., 1995; Marcus, Waugh & Westervelt, 1990).
- This model is not very sensitive to the model parameters and the connection weight function. The parameters can be chosen in a very wide range. The weight function can be any monotonically decreasing function.

ACKNOWLEDGMENTS

This work was supported by Natural Sciences and Engineering Research Council (NSERC) and Materials and Manufacturing Ontario (MMO) of Canada.

REFERENCES

Al-Sultan, K. S. & Aliyu, D. S. (1996). A new potential field-based algorithm for path planning. *J. of Intelligent and Robotic Systems*, 17(3), 265-282.

Beom, H. R. & Cho, H. S. (1995). Sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning. *IEEE Trans. on Systems, Man, and Cybernetics*, 25(3), 464-477.

Chang, C. C. & Song, K. T. (1997). Environment prediction for a mobile robot in a dynamic environment. *IEEE Trans. on Robotics and Automation*, 13(6), 862-872.

Gaudiano, P., Zalama, E. & Lopez Coronado, J. (1996). An unsupervised neural network for low-level control of a mobile robot: Noise resistance, stability, and hardware implementation. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 26(3), 485-496.

Glasius, R., Komoda, A. & Gielen, S. C. A. M. (1994). Population coding in a neural net for trajectory formation. *Network: Computation in Neural Systems*, 5, 549-563.

Glasius, R., Komoda, A. & Gielen, S. C. A. M. (1995). Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1), 125-133.

Grossberg, S. (1973). Contour enhancement, short-term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, 52, 217-257.

Grossberg, S. (1982). *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control.* Boston: Reidel Press.

Grossberg, S. (1983). Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Trans. Systems, Man, and Cybernetics*, 13(5), 815-926.

Grossberg, S. (1988). Non-linear neural networks: Principles, mechanisms, and architecture. *Neural Networks*, 1, 17-61.

Hodgkin, A. L. & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology London,* 117, 500-544.

Ilari, J. & Torras, C. (1990). 2nd path planning: A configuration space heuristic approach. *International Journal of Robotics Research*, 9(1), 75-91.

Li, L. & Ogmen, H. (1994). Visually guided motor control: Adaptive Sensorimotor mapping with on-line visual-error correction. In: *Proceedings of the World Congress on Neural Networks.* 127-134.

Li, Z. X. & Bui, T. D. (1998). Robot path planning using fluid model. *Journal of Intelligent and Robotic Systems*, 21, 29-50.

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *IEEE Trans. Computers, 32*, 108-320.

Marcus, C. M., Waugh, F. R. & Westervelt, R. M. (1990). Associative memory in an analog iterated-map neural network. *Physical Review A*, 41(6), 3355-3364.

Ogmen, H. & Gagne, S. (1990a). Neural models for sustained and on-off units of insect lamina. *Biological Cybernetics,* 63, 51-60.

Ogmen, H. & Gagne, S. (1990b). Neural network architecture for motion perception and elementary motion detection in the fly visual system. *Neural Networks,* 3, 487-505.

Ong, C. J. & Gilbert, E. G. (1998). Robot path planning with penetration growth distance. *Journal of Robotic Systems*, 15(2), 57-74.

Yang, S. X. & Meng, M. (2000a). Neural network approach to real-time collision-free motion planning. *Neural Networks,* 13(2), 133-148.

Yang, S. X. & Meng, M. (2000b). An efficient neural network method for real-time motion planning with safety consideration. *Robotics and Autonomous Systems,* 32(2-3), 115-128.

Yang, S. X. & Meng, M. (2001). Neural network approaches to dynamic collision-free trajectory generation. *IEEE Trans. on Systems, Man, and Cybernetics, Part B,* 31 (3), 302-318.

Yang, X. (1999). *Neural Network Approaches to Real-Time Motion Planning and Control of Robotic Systems.* PhD dissertation, University of Alberta, Canada.

Zalama, E., Gaudiano, P. & Lopez Coronado, J. (1995). A real-time, unsupervised neural network for the low-level control of a mobile robot in a non-stationary environment. *Neural Networks,* 8, 103-123.

Zelinsky, A. (1994). Using path transforms to guide the search for findpath in 2nd *International Journal of Robotics Research*, 13(4), 315-325.

SECTION II:

HYBRID EVOLUTIONARY SYSTEMS FOR MODELLING, CONTROL AND ROBOTICS APPLICATIONS

Chapter V

Evolutionary Learning of Fuzzy Control in Robot-Soccer

P.J.Thomas and R.J.Stonier
Central Queensland University, Australia

ABSTRACT

In this chapter an evolutionary algorithm is developed to learn a fuzzy knowledge base for the control of a soccer micro-robot from any configuration belonging to a grid of initial configurations, to hit the ball along the ball to goal line of sight. A relative coordinate system is used. Forward and reverse mode of the robot and its physical dimensions are incorporated, as well as special considerations to cases when in its initial configuration, the robot is touching the ball.

INTRODUCTION

An important aspect of fuzzy logic application is the determination of a fuzzy logic knowledge base to satisfactorily control the specified system, whether this is derivable from an appropriate mathematical model or just from system input-output data. Inherent in this are two main problems. The first is to obtain an adequate knowledge base (KB) for the controller, usually obtained from expert knowledge, and second is that of selection of key parameters defined in the method.

The KB is typically generated by expert knowledge but a fundamental weakness with this static acquisition is that it is frequently incomplete, and its control strategies are conservative. To overcome this one approach is to construct self-organising fuzzy logic controllers (Yan, 1994). These self-organising fuzzy logic controllers are used mainly for the creation and modification of the rule base. Of interest is the question of how this self-organisation and adaptation can be carried out in an automated fashion. One way is to incorporate genetic/evolutionary algorithms to form genetic fuzzy systems, (Karr, 1991; Thrift, 1991; Cordòn, 1995).

Evolutionary learning of fuzzy controllers in a three-level hierarchical, fuzzy logic system to solve a collision-avoidance problem in a simulated two-robot system is discussed in Mohammadian (1998a). A key issue is that of learning knowledge in a given layer sufficient for use in higher layers. We need to find a KB that is effective, to some acceptable measure, in controlling the robot to its target from 'any' initial configuration. One way is to first learn a set of local fuzzy controllers, each KB learned by an evolutionary algorithm from a given initial configuration within a set of initial configurations spread uniformly over the configuration space. These KBs can then be *fused* through a *fuzzy amalgamation* process (Mohammadian, 1994, 1998b; Stonier, 1995a, 1995b), into the global (final), fuzzy control knowledge base. An alternative approach (Mohammadian, 1996; Stonier, 1998), is to develop an evolutionary algorithm to learn directly the 'final' KB by itself over the region of initial configurations.

In this chapter we use this latter approach and incorporate special attributes to cover the difficult cases for control when the robot is close and touching the ball. A relative coordinate system is used and terms are introduced into the fitness evaluations that allow both forward and reverse motion of the soccer robot. We define the robot soccer system, the design of the fuzzy controller, the design of the evolutionary algorithm and finish with a short presentation of results for control of the robot from a far distance from the ball and from configurations close and touching the ball.

ROBOT SOCCER SYSTEM

The basic robot soccer system considered is that defined for the Federation of Robot-Soccer Association, Robot World Cup (*www.fira.net*). All calculations for vision data processing, strategies and position control of the robots are performed on a centralised host computer. Full specifications of hardware, software and basic robot strategies that are employed in this type of micro-robot soccer system can be found in Kim (1998).

Kinematics

The kinematics of a wheelchair-style robot is given by Equation 1 from Jung (1999).

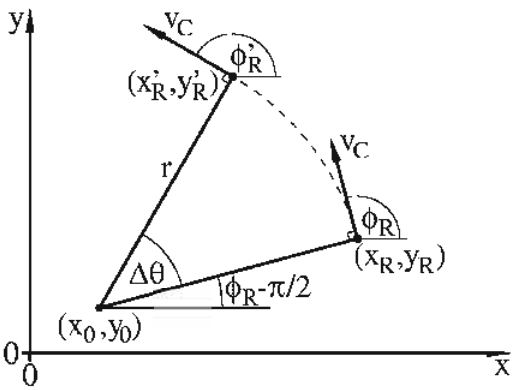
$$\begin{bmatrix} v_C \\ \omega \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ -1/L & 1/L \end{bmatrix} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \tag{1}$$

where v_L is the instantaneous speed at the left wheel of the robot, v_R is the instantaneous speed at the right wheel of the robot, L is the wheel base length, v_C is the instantaneous speed of the robot centre, ω is the instantaneous angular speed about the instantaneous point of rotation (x_0, y_0). The radius of the arc r is determined from $v_C = r\,\omega$, which is the distance between (x_0, y_0) and v_C.

Let the next robot position be approximated by a small time interval Δt. Assume v_L and v_R are constant over this interval. If $\omega = 0$, the robot is moving in a straight line. Equation 2 gives the next robot position using linear displacement $\Delta s = v_C \Delta t$:

$$\begin{bmatrix} x_R' \\ y_R' \\ \phi_R' \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \\ 0 \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\phi_R) \\ \Delta s \sin(\phi_R) \\ \phi_R \end{bmatrix} \tag{2}$$

Figure 1: Curvilinear formulae symbols



When $\omega \neq 0$, the robot scribes an arc. Curvilinear robot paths are calculated using translation, rotation and translation Equation 3. Refer to Figure 1 for the following derivation:

First, determine the point of rotation (x_0, y_0):

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \end{bmatrix} - r \begin{bmatrix} \cos(\phi_R - \pi/2) \\ \sin(\phi_R - \pi/2) \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \end{bmatrix} + r \begin{bmatrix} -\sin(\phi_R) \\ \cos(\phi_R) \end{bmatrix}$$

Translate point of rotation to origin:

$$\begin{bmatrix} x_R^1 \\ y_R^1 \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = r \begin{bmatrix} \sin(\phi_R) \\ -\cos(\phi_R) \end{bmatrix}$$

Rotate about the z-axis (counter-clockwise positive):

$$\begin{bmatrix} x_R^2 \\ y_R^2 \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta) & \sin(\Delta\theta) \\ -\sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix} \begin{bmatrix} x_R^1 \\ y_R^1 \end{bmatrix} = r \begin{bmatrix} \sin(\phi_R + \Delta\theta) \\ -\cos(\phi_R + \Delta\theta) \end{bmatrix}$$

Translate origin to point of rotation:

$$\begin{bmatrix} x_R' \\ y_R' \end{bmatrix} = \begin{bmatrix} x_R^2 \\ y_R^2 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \end{bmatrix} + r \begin{bmatrix} \sin(\phi_R + \Delta\theta) - \sin(\phi_R) \\ -\cos(\phi_R + \Delta\theta) + \cos(\phi_R) \end{bmatrix}$$

Finish off by adding in robot angle correction:

$$\begin{bmatrix} x_R' \\ y_R' \\ \phi_R' \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \\ \phi_R \end{bmatrix} + \begin{bmatrix} \sin(\phi_R + \Delta\theta) & -\sin(\phi_R) & 0 \\ -\cos(\phi_R + \Delta\theta) & \cos(\phi_R) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ r \\ \Delta\theta \end{bmatrix} \quad (3)$$

where $\phi'_R \in [0,2\pi)$ is necessarily constrained for input into the fuzzy system. The following parameter values were used: $L = 68.5(mm), \Delta t = 1/60(s)\ \Delta t = 1/60(s)$. The playable region was defined as a rectangle from coordinate (0,0) to (1500,1300) (measurements in mm.), ignoring the goal box at each end of the field.

FUZZY CONTROL SYSTEM DESIGN

The discussion on kinematics above shows, excluding momentum and friction, that only two variables, the velocity of the left and right wheels, $y_1 = v_L$ and $y_2 = v_R$, control the motion of the robot. These two variables are taken as output of the fuzzy control system now described below. Input variables for the fuzzy control system are taken to be the position of the robot relative to the ball, described by $n = 3$ variables $x_1 = d^2$, $x_2 = \theta$ and $x_3 = \phi$ as shown in Figure 2.

These relative coordinates were used in preference to Cartesian coordinate variables for a number of reasons, one being that it reduced the number of rules in the fuzzy KB. Distance squared was used to reduce the calculation cost by not using a square root function, effectively applying a "more or less" hedge. The angle of the robot relative to the ball goal line was used instead of the ball robot line because of positional error arising from image capture pixel size in determining the position of each object. The vision system has an inherent $\pm 4.5\,mm$ error caused by pixel size. The pixel size error causes the angle of the line error to be inversely proportional to the distance between the points used to calculate the line. However, one of the points used to calculate the $\overline{BG}$ line is at the centre of the goal line. The allowable angle range when close to the goal offsets the error caused in determining the line. The vision system error has negligible effect on placing the ball into the goal when using $\overline{BG}$ as a reference.

Figure 2: Relative coordinate parameters

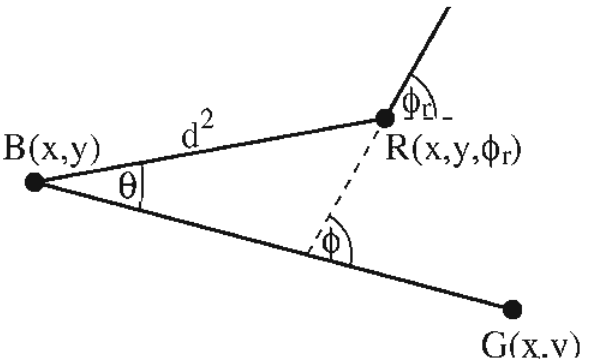


Figure 3: Fuzzy input sets

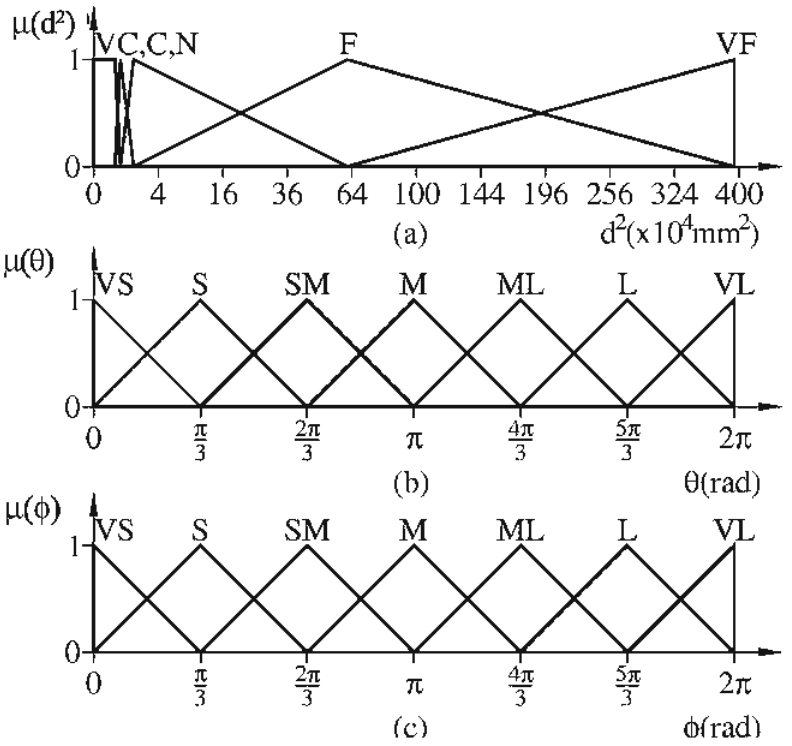


Figure 3 shows the fuzzy input sets used to define the "attack ball strategy." For all rules seven sets are defined for both angles θ and ϕ : **VS** is Very Small, **S** is Small, **SM** is Small Medium, **M** is Medium, **ML** is Medium Large, **L** is Large and **VL** is Very Large. Five sets are defined for distance squared: **VC** is Very Close, **C** is Close, **N** is Near, **F** is Far and **VF** is Very Far.

The values of y_1 and y_2 are taken to be integers lying in the interval [-128, 127]. We take 256 B_k output fuzzy sets each corresponding to centre $\bar{y}_k = -128 + (k-1)$ for $k = 1,\ldots,256$. In this case the name of the sets are the same as the output centres $\bar{y}_k$ of the sets.

The purpose of taking 256 B_k output fuzzy sets instead of 255, $B_k \in$ [-127, 127], is a technical issue to allow the use of a binary mutation operator in the evolutionary algorithm. The velocities are in fact capped to $v_L, v_R \in$ [-127, 127] before transmission to the robots.

Taking a large number of output sets serves three purposes:

(i) It does not affect the computational cost of the fuzzy controller; the solution can be as fine as it needs to be.

(ii) The $\bar{y}_k$ are the control values used for the left and right wheel motors - eliminating conversion.
(iii) It reduces erratic behaviour of the evolutionary algorithm (finer control) during mutation.

There were $7 \times 7 \times 5 = 245$ rules in a complete fuzzy knowledge base for this system. In general, the j^{th} fuzzy rule has the form:

If (x_1 is A_1^j and x_2 is A_2^j and x_3 is A_3^j)

Then (y_1 is B_1^j and y_2 is B_2^j)

where A_k^j, $k = 1,2,3$ are normalised fuzzy sets for input variables x_k, $k = 1,2,3$, and where B_m^j, $m = 1,2$ are normalised fuzzy sets for output variables y_m, $m = 1,2$.

Given a fuzzy rule base with M rules, a fuzzy controller as given in Equation 4 uses a singleton fuzzifier, Mamdani product inference engine and centre average defuzzifier to determine output variables:

$$y_k = \frac{\sum_{j=1}^{M} \bar{y}_k^j \prod_{i=1}^{n} \mu_{A_i^j}(x_i)}{\sum_{j=1}^{M} \prod_{i=1}^{n} \mu_{A_i^j}(x_i)} \qquad (4)$$

where $\bar{y}_k^j$ are centres of the output sets B_k^j.

These values, 490 of them, are typically unknown and require determination in establishing valid output for controls to each wheel of the robot. Since there is lack of *a priori* knowledge about the system control, we used evolutionary algorithms (EAs) (Michalewicz, 1994) to search for an acceptable solution.

EVOLUTIONARY LEARNING

Our objective here is to learn a rule base for the control of this system. The first problem is how to formulate the knowledge base as a string in the population.

Each output fuzzy set is represented by an integer in the interval [-128, 127]. We can form an individual string $\underset{\sim}{P}$ as a string of $2M = 490$ consequents (integers under the identification above):

$$\underline{s} = \left\{ s_1^1, s_2^1, \ldots, s_1^k, s_2^k, \ldots, s_1^M, s_2^M, \right\}$$

where $s_j, j = 1, 2$ is an integer in the interval [-128,127].

The population at generation t, $P(t) = \underline{s}^n : n = 1, \ldots, N$, where N is the number of individuals in the population. The population at the next generation $P(t = 1)$ was built using a full replacement policy. Tournament selection, with n_T being the tournament size, determined two parent strings for mating in the current generation. Geometric crossover with probability p_c was used for generating two child strings from the parent strings, for insertion in the next generation's population. In each string, the integer components were stored as two's complement byte-sized quantities, and binary mutation was undertaken on each string in the new population with probability p_m. (Elitism was not used, for it was found to cause premature convergence of the algorithm.)

Fitness evaluation of each individual was calculated by scribing a path using the fuzzy controller and stopping when either:

(i) iteration (time steps) reached a prescribed limit (100), or
(ii) the path exceeded the maximum allowable distance from the ball, or
(iii) the robot collides with the ball.

In (iii) care needs to be taken recognising the finite size of the robot. The robot is a square with size of 80 *mm* and the ball has a diameter of 42.7 *mm*. Detecting a collision of the robot and ball is made by calculating the distance of the ball centre $(x_B, y_B) = (750, 650)$ perpendicular to the line in the direction of the robot d_{NL} passing through the centre of the updated position of the robot (x'_R, y'_R), and the distance of the ball d_{AL} projected onto that line. These values are determined as:

$$d_{AL} = \frac{|(x_B - x'_R) + m(y_B - y'_R)|}{\sqrt{m^2 + 1}}$$

$$d_{NL} = \frac{|(y_B - y'_R) - m(x_B - x'_R)|}{\sqrt{m^2 + 1}}$$

where m is the gradient of the line passing through the robot centre, in the direction of the robot. The following quantity is used to define an exclusion region determined by the physical size of the robot:

A flag "HitBall" is raised when the following condition is true:

$$\text{IF } (((d_{NL} < 40) \text{ AND } (d_{AL} < 61.35)) \text{ OR } ((d_{NL} < 61.35)$$
$$\text{AND } (d_{AL} < 40)) \text{ OR } (r^2_{corner} < 61.35^2)) \text{ THEN (HitBall=TRUE)}$$

where, $r^2_{corner} = (d_{AL} - 40)^2 + (d_{NL} - 40)^2$.

The final position of the path was used to evaluate the fitness of each individual as given by Equation 5:

$$\sum_{C} (T_1 + T_2 + T_3 + T_4) \tag{5}$$

The fitness is calculated as a sum of a number of different quantities, over a set C of initial starting configurations, each configuration specifying robot coordinates x_R, y_R, and angle ϕ describing the orientation of the robot relative to the $\overline{BG}$ line.

There were 273 initial configurations. The first 28 are defined with the robot touching the ball on each of its four sides in seven different orientations θ around the ball. The remaining initial configurations were defined with seven θ angles on five distance rings from the ball with seven ϕ angles.

The first 28 configurations $c \in [0, 28)$, are given by:

$$x_R = x_B + 61.35\cos(\theta),\ y_R = y_B + 61.35\sin(\theta), \text{ and } \phi \text{ with:}$$

$$\theta \in \{0, \pi/3, 2\pi/3, \pi, 4\pi/3, 5\pi/3, 71\pi/36\}, \text{and } \phi \in \{0, \pi/2, \pi, 3\pi/2\}.$$

The remaining initial configurations $c \in [28, 273)$, are given by:

$$x_R = x_B + d\cos(\theta),\ y_R = y_B + d\sin(\theta), \text{ and } \phi \text{ with:}$$

$$d \in \{77.92, 122.75, 310.4, 784.94, 1900\},$$
$$\theta \in \{0, \pi/3, 2\pi/3, \pi, 4\pi/3, 5\pi/3, 71\pi/36\},$$
$$\phi \in \{0, \pi/3, 2\pi/3, \pi, 4\pi/3, 5\pi/3, 71\pi/36\}.$$

The first quantity in the fitness sum is $T_1 = d^2(R, DP)$. It is the final squared distance between the robot centre R and the destination point $DP = (688.5, 650)$ when the path is terminated as described above. The term is used to determine accuracy of the fuzzy controller to control the system to the desired destination configuration.

The second term T_2 is the iteration count for each path. This quantity is used to minimise the time taken to reach the desired destination configuration.

The third quantity is $T_3 = 1000\sin^2(\phi)$ where ϕ is the final angle of the robot relative to line $\overline{BG}$. This term is included to enable forward facing and reverse facing solutions to be accepted at the final destination.

The fourth quantity T_4 is a penalty function that is only applied for those configurations $c \in [0, 28)$. It is described in Equation 6:

$$T_4 = \begin{cases} 10000 & \textit{if} \quad \theta \in [11\pi/12, 13\pi/12) \quad \textit{and} \quad \sin^2(\phi) > 0.25 \\ 10000 & \textit{if} \quad \theta \notin [11\pi/12, 13\pi/12) \\ 0 & \textit{otherwise} \end{cases} \tag{6}$$

It is a constant penalty used to drive the solution away from paths that hit the ball when considering the first 28 initial configurations. Without this penalty, the best solutions obtained via evolutionary learning are invariably ones that try to run through the ball.

The evolutionary algorithm was terminated after a prescribed number of generations. The best individual, that is, the one with the minimum fitness, is taken as the "best" fuzzy logic controller determined by the algorithm for this problem.

RESULTS

The evolutionary algorithm was found to easily learn a fuzzy controller for when fitness was evaluated for a single initial configuration.

Establishing learning over a set of initial configurations from $c = 0$ to $c = 28$ where the robot was placed in contact with the ball was difficult; appropriate sets of evolutionary parameters needed to be defined with a mutation schedule to ensure diversity in the population at different stages in the learning, for example after every 1,000 generations. The reason for the difficulty was that the algorithm tended to lock fuzzy control into always forward or always reverse motion of the robot, with the consequence that not the shortest distance path was achieved, and invariable penalty constraints were broken.

Learning the fuzzy control over the set of all configurations incorporated the difficulties that had to be overcome for those configurations close to the ball. The algorithm tended to lock into local minima when considering multiple configurations. The local minima existed due to the algorithm finding a good single path amongst the many that influenced nearby paths.

Typical values in simulations were: the size of the population $N = 200$, probability of crossover $p_c = 0.6$ and the number of tournament contestants $n_T = 8$. Mutation probability was defined as a schedule: $p_m = 0.05 - 0.00048(gen \bmod (1000))$, which decreased mutation with increasing generation number and recommenced with high mutation every 1,000 generations. The evolutionary algorithm was usually run for batches of 10,000 generations.

Due to limited space we present here a few results obtained from the evolutionary learning of the knowledge base, two examples showing the control of the robot from a large distance from the ball, and two showing control of the robot touching the ball.

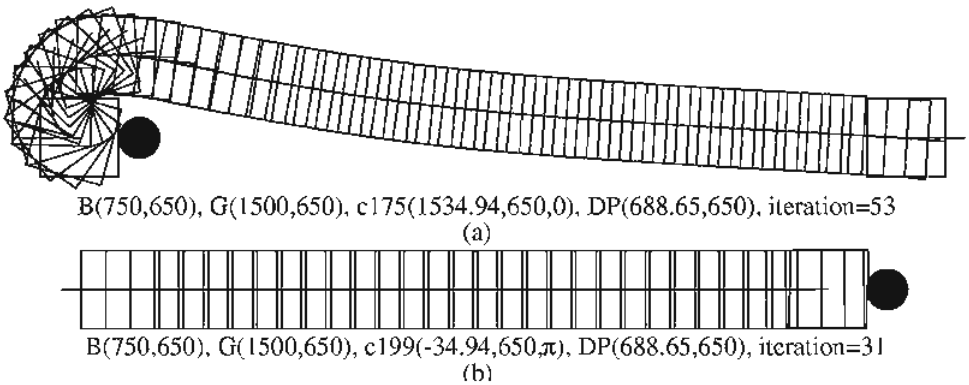
Learning of the Fuzzy Control at a Distance

Figure 4(a) shows the path from initial configuration $c = 175$. For this initial configuration the robot is placed to the far right of the ball and on the ball to goal line. It took 53 time steps to reach the destination point DP with a final angle of ϕ (rad).

In Figure 4(b) the path from initial configuration $c = 199$, with the robot to the far left, facing away from the ball on the ball to goal line, took just 31 time steps to reach the destination point DP with a final angle of ϕ (rad). Note that in both cases the robot approached the destination in reverse mode.

These graphs are typical of the fuzzy control of the robot starting from initial positions at a "large" distance from the ball. Destination and final angle accuracy was excellent. Evolutionary learning was quite rapid, with acceptable solutions resulting in smooth paths to the destination started appearing within a few hundred generations. Further learning resulted in faster control to the destination.

Figure 4: Long distance paths



Learning of Fuzzy Control Close to the Ball

Learning of the fuzzy control of the robot close to the ball was more difficult. Figure 5(a) shows the path from initial configuration $c = 1$ which took 19 time steps to reach the destination point *DP* with final angle of 0 (*rad*). In Figure 5(b) the path from the initial configuration $c = 13$ took 24 time steps to reach the destination point *DP* with a final angle of 0 (*rad*). In both cases the robot approached the final destination in forward mode.

The starting initial configurations in which the robot was touching the ball were the most difficult to learn, for they were responsible for the majority of the penalty function evaluations in the fitness calculations for each individual of the evolutionary algorithm. The hardest initial configuration to learn was $c = 13$.

COMMENTS

This chapter detailed the learning of a fuzzy logic controller to control a soccer robot to a point behind the ball in the direction of the goal by using an evolutionary algorithm. Learning of a single robot path was very easy. However, learning of several paths form different initial configurations caused many difficulties.

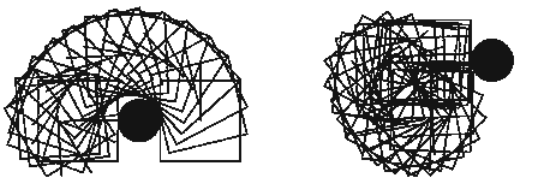
Several starting configuration evaluations caused the final approach of all paths to be either forward or reverse facing. To achieve the final approach heading, the evolutionary algorithm learnt to use chatter and high momentum turns. If a restriction on turning was applied, the algorithm learnt to execute low momentum turns.

The cause of these difficulties was identified as:

(i) Insufficient number of inputs to the fuzzy system, the rule base could not cater for all of the information need to control the robot to forward and reverse facing solutions.

(ii) Multi-criteria optimisation problems caused by summing all terms from all path evaluations forming the fitness value.

Figure 5: Short distance paths



c001(811.35,650,π/2), iteration=19 (a)

c013(688.65,650,π/2), iteration=24 (b)

B(750,650), G(1500,650), DP(688.65,650)

NEW RESEARCH

The research presented here has since been extended and a brief description is given. The full analysis and results will be published elsewhere.

The number of input variables for the given problem was extended to include left wheel and right wheel velocities, with the number of membership sets for each variable extended to 7, yielding $7^5 = 16807$ rules in the complete fuzzy knowledge base.

The output variables were changed from left and right wheel velocities to changes in these velocities, namely: Δv_L and Δv_R, with final wheel velocities $v'_L = v_L + \Delta v_L$ and $v'_R = v_R + \Delta v_R$. The number of output member sets was reduced to eight $B_k \in [-28, 28]$, $k = 0, \ldots, 7$.

The number of initial configuration was increased to a grid: $x = -750 + 100(k - 1)$ for $k = 1, \ldots, 31$ and $y = -650 + 100(k - 1)$ for $k = 1, \ldots, 27$ excluding the ball position. Each grid point has five angles: $\theta = 2(k-1)\pi / 5$ for $k = 1, \ldots, 5$. The total number of initial configurations is therefore $C = 5(31 \times 27 - 1) = 4180$. All initial configurations start with zero, left and right, wheel velocity.

Each output fuzzy set is represented by an integer in the interval [0, 7]. An individual string $\underset{\sim}{P}$ is now of length $2M = 33614$ consequents: $\underset{\sim}{s} = \{s_1^1, s_2^1, \ldots, s_1^k, s_2^k, \ldots, s_1^M, s_2^M\}$ where s_j, $j = 1, 2$ is an integer in the interval [0, 7].

Evolutionary learning was again used with a population of size $N = 2000$, full replacement policy, tournament selection with size $n_T = 3$ and one point crossover with probability $p_c = 0.6$. Elitism was now used, with the 10 best individuals carried from the old population to the new population. An incremental mutation operator with probability $p_m = 0.01$ replaced the binary mutation used previously. This mutation operator increments/decrements s_k by one with equal probability and has boundary checking, that is, if $s_k = 0$, it was incremented to $s_k = 1$, and if $s_k = 7$, it was decremented to $s_k = 6$.

The final position of the path was again used to evaluate the fitness of each individual as given by Equation 7:

$$\sum_C (\alpha_1 T_1 + \alpha_2 T_2 + \alpha_3 T_3 + \alpha_4 T_4) \tag{7}$$

where $\alpha_1 = 1.0$, $\alpha_2 = 1.0$, $\alpha_3 = 100.0$, $\alpha_4 = 0.0$. The weight coefficients for the first two terms were equal, the terminal angle coefficients was heavily weighted and constant penalty was turned off.

A new learning for the algorithm was implemented as follows. The evolutionary algorithm was run sequentially through the full number of initial configurations, being allowed to run for 10 generations at each configuration before moving to the next. It was stopped after a total of 500,000 generations in all were completed.

The results obtained in the final "best" fuzzy knowledge were excellent, obtaining very smooth continuous paths to the target with both forward and reverse facing in the final position depending on the initial configuration. Only a very small number of aberrations existed, but the paths to the target were still acceptable. Due

Figure 6: Long distant path

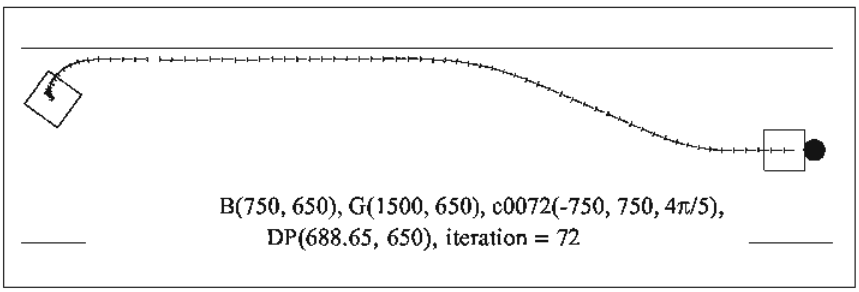
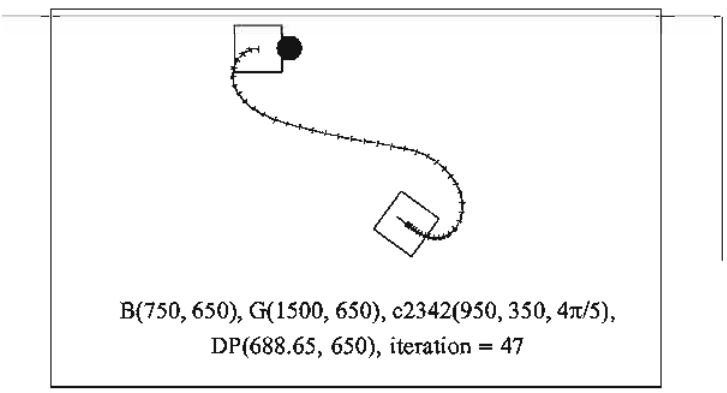


Figure 7: Short distant path



to limited reporting space, we show only two of the many images obtained in Figures 6 and 7. Note one has final approach to the ball forward facing, the other reverse facing; one is from a far distance and one is close to the ball.

This research is being further extended for initial input left and right wheel velocities lying in the full range of admissible values.

REFERENCES

Cordòn, O. & Herrera, F. (1995). A general study on genetic fuzzy systems. In Winter, G & Cuesta, P. (Eds.), *Genetic Algorithms in Engineering and Computer Science,* John Wiley and Sons, 33-57.

Jung, M.-J., Shim, H.-S., Kim, H.-S. & Kim, J.-H. (1999). The omni-directional mobile robot OmniKity-Y(OK-I) for RoboSOT category. In Stonier, R.J. & Kim, J.H. (Eds) *Proceedings of the FIRA Robot World Cup 1998*, 37-42.

Karr, C.L. (1991). Applying genetics. *AI Expert*, 38-43.

Kim, J.H. (1998). *Lecture Notes on Multi-Robot Cooperative System Development.* Green Publishing Co.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs* (2nd Edition). New York: Springer Verlag.

Mohammadian, M. (1998). *Genetic learning of fuzzy control rules in hierarchical and multi-layer fuzzy logic systems with application to mobile robots,* PhD Thesis, Central Queensland University, Rockhampton, Australia.

Mohammadian, M. & Stonier, R.J. (1994). Generating fuzzy rules by genetic algorithms. *Proceedings of 3rd International Workshop of Robot and Human Communication*, Nagoya, Japan, 362-367.

Mohammadian, M. & Stonier, R.J. (1996). Fuzzy rule generation by genetic learning for target tracking. *Proceedings of the 5th International Intelligent Systems Conference*, Reno, Nevada, 10-14.

Mohammadian, M. & Stonier, R.J. (1998). Hierarchical fuzzy control, *Proceedings of the 7th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Paris, 621-629.

Stonier, R.J. (1995). Adaptive learning using genetic algorithms and evolutionary programming in robotic systems, *First Korea-Australia Joint Workshop on Evolutionary Computing*, 183-198.

Stonier, R.J. & Mohammadian, M. (1995). Self learning hierarchical fuzzy logic controller in multi-robot systems. *Proceedings of the IEA Conference Control95*, Melbourne, Australia, 381-386.

Stonier, R.J. & Mohammadian, M. (1998). Knowledge acquisition for target capture, *Proceedings of the International Conference on Evolutionary Computing ICEC'98*, Anchorage, Alaska, 721-726.

Thrift, P. (1991). Fuzzy logic synthesis with genetic algorithms. *Proceedings of the 4th International Conference on Genetic Algorithms*, 509-513.

Yan, J., Ryan, M. & Power, J. (1994). *Using Fuzzy Logic*, New York: Prentice Hall.

Chapter VI

Evolutionary Learning of a Box-Pushing Controller

Pieter Spronck, Ida Sprinkhuizen-Kuyper, Eric Postma and Rens Kortmann
Universiteit Maastricht, The Netherlands

Abstract

In our research we use evolutionary algorithms to evolve robot controllers for executing elementary behaviours. This chapter focuses on the behaviour of pushing a box between two walls. The main research question addressed in this chapter is: how can a neural network learn to control the box-pushing task using evolutionary-computation techniques? In answering this question we study the following three characteristics by means of simulation experiments: (1) the fitness function, (2) the neural network topology and (3) the parameters of the evolutionary algorithm. We find that appropriate choices for these characteristics are: (1) a global external fitness function, (2) a recurrent neural network, and (3) a regular evolutionary algorithm augmented with the doping technique in which the initial population is supplied with a solution to a hard task instance. We conclude by stating that our findings on the relatively simple box-pushing behaviour form a good starting point for the evolutionary learning of more complex behaviours.

Introduction

Imagine a cat on a rooftop. It has just spotted a juicy pigeon sitting on a window sill and is wondering how to catch this prey. The situation is tricky: there are two routes the cat can take, both of them involving walking on narrow ledges and requiring daring tricks of balance. The cat decides to take the shortest route and tries to lower itself onto a ledge beneath. While trying to do so, it notices that the chance of toppling over and falling three stories down onto a busy shopping street is becoming increasingly more realistic. The cat now decides to abandon its plan and sets its senses to something more practical.

From a traditional Artificial Intelligence point of view, this navigation problem is not that difficult. The start and goal positions are known, the possible routes are clear, and apparently, the cat has developed a plan to catch the bird. However, the successful execution of the plan critically depends on the cat's low-level interactions with its environment, rather than its high-level planning capabilities. Hitherto, the Artificial Intelligence community has given little attention to low-level control mechanisms (e.g., equilibrium controllers) as compared to high-level controllers (e.g., symbolic problem-solving systems).

Low-level controllers are typically needed for autonomous systems dealing with elementary tasks in dynamic partially observable environments. They form the foundation of high-level controllers executing more complex tasks in the environment (Brooks, 1986). In this chapter we focus on the elementary task of pushing a box between two walls. The box-pushing task was originally introduced (albeit in a slightly different form) by Lee, Hallam and Lund (1997). Pushing an object is an important aspect of robot soccer, a modern platform for autonomous systems research (Asada & Kitano, 1999), and underlies many more complex behaviours such as target following, navigation and object manipulation. While the deceptively simple task of pushing an object is usually disregarded in favour of the seemingly more challenging task of determining a strategic position, pushing is far from trivial and deserves at least as much attention as the strategic task.

The main research question addressed in this chapter is: how can a neural network learn to control the box-pushing task using evolutionary-computation techniques? In answering this question we study the following three characteristics by means of simulation experiments: (1) the fitness function, (2) the neural network topology and (3) the parameters of the evolutionary algorithm.

The outline of the remainder of this chapter is as follows. First, we discuss some background on the use of neural networks and evolutionary algorithms in learning to control a robot. Then we describe the goal of the research in terms of the three characteristics discussed above (i.e., the fitness function, the neural-network

topology and the parameters of the evolutionary algorithm). We give an overview of the standard procedure used in our simulation experiments, followed by a presentation and discussion of the results. Finally, we draw conclusions.

Background

This section provides some background on the approach pursued in our experiments by discussing the use of neural and evolutionary computation techniques in controlling an autonomous robot.

Neural networks offer useful models for learning to control autonomous robots. Although there exist many effective learning algorithms for automatically setting the weights of the network, most algorithms require a carefully prepared set of training examples consisting of pairs of input and desired-output patterns. For freely moving robots in a semi-realistic environment, the preparation of a training set is rather tedious. It is not uncommon that a set of training examples cannot be generated at all. For instance, if the environment in which the controller has to work is (partially) unknown, a training set cannot take into account all the situations that the controller may encounter.

An alternative way to determine the neural network weights is by employing evolutionary algorithms (Bäck, 1996; Yao, 1995). Evolutionary algorithms have many advantages over regular training methods especially for controlling robots (Arkin, 1998). Besides the fact that evolutionary algorithms offer the ability to learn both the weight values and the topology (whereas regular training methods often are limited to determining only the weight values), the only requirement for evolutionary algorithms to work is the availability of a so-called fitness function. A fitness function is an evaluation function indicating the relative success of one solution compared to all the others. Such a fitness function can be defined on a set of examples, comparable to the training set used in most regular training methods. Alternatively, the fitness function can be defined as the quality of the actual behaviour of a neural-network-controlled robot during a test run. This last form of evolutionary learning is called *Genetic Reinforcement Learning* and was relatively unknown until Darrell Whitley (1993) introduced it in his experiments with the GENITOR algorithm.

The main disadvantage of evolutionary algorithms is their inherent unpredictability with respect to the computation time and the quality of the final solution found. It is very hard to predict the time before the algorithm comes up with a satisfactory solution to the problem at hand, and it is often difficult to judge whether significantly better solutions are possible (Goldberg, 1989). In view of these limitations, we

decided to study the application of evolutionary algorithms to a relatively simple box-pushing task extensively in order to find an optimal configuration for the neural controller and the evolutionary algorithm.

Goal

The goal of our study is to find an optimal design of the evolutionary experiments in order to optimise the quality of a box-pushing controller. In particular, our aim is to increase our understanding of how to set up the experiments in an optimal way so that we can be more effective in designing neural controllers for more complex tasks.

In the box-pushing task, the robot is faced with the task of pushing a box as far as possible between two walls in a limited period of time. The inputs of the neural network are the signals received by the robot's proximity sensors. The output of the network rotates the wheels of the robot.

As exemplified in our research question, the following three questions of the experimental set-up are addressed in our research.

1. *What is a suitable fitness function for the evolutionary algorithm?* The fitness function is a measurement of the success of a solution proposed by the evolutionary algorithm, and is the single most important aspect of any evolutionary system.
2. *What is an appropriate neural network topology to solve the box-pushing task?* The main candidates for the neural network are the feedforward and recurrent network topologies. The main advantage of a feedforward network is that it is quick and simple to evolve. The main advantage of a recurrent network is that it has the ability to store and recall values from previous cycles of the robot run.
3. *What is a proper choice for the parameters of the evolutionary algorithm?*

In the next section, we discuss the experimental procedure followed in answering these questions.

Experimental Procedure

The Robot

The robot we used in our studies is of the Khepera type (Mondada et al., 1993). For this robot a good simulation is publicly available. Controllers developed with this simulation have shown to work well in practice. We used the simulator on

Figure 1: Screenshot from the Unix Khepera simulator. To the left the robot world, with the walls (rectangular blocks), the robot (grey circle) and the box (black circle). The small black spots indicate the starting positions used for the robot (the lower three spots) and the box (the upper three spots).

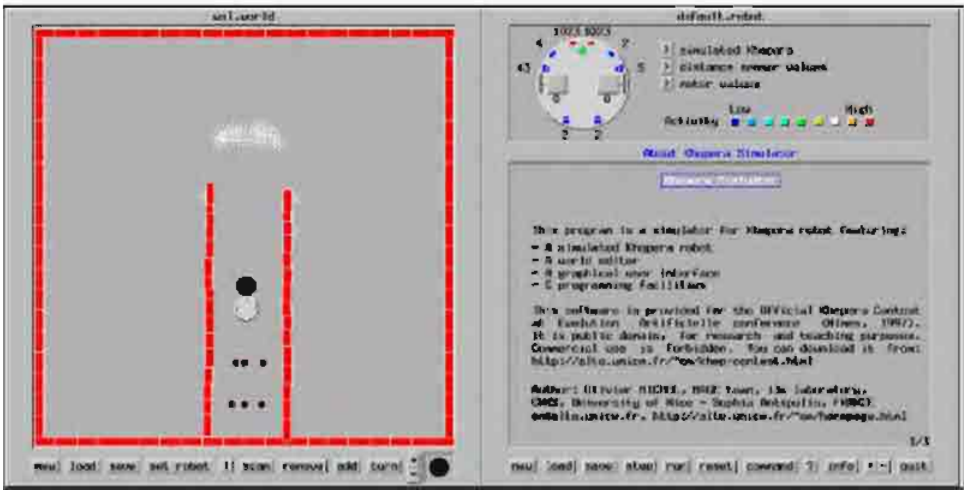
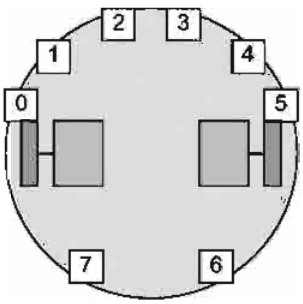


Figure 2: Overview of the Khepera robot



Neural controller inputs:

0 to 7:	Sensor 0 to 7 distance values.
8:	Sensor 0 - 1
9:	Sensor 1 - 2
10:	Sensor 2 - 3
11:	Sensor 3 - 4
12:	Sensor 4 - 5
13:	Sensor 6 - 7

Inputs 8 to 13 are the edge detectors.

its original Unix-platform (Figure 1) and also ported it to a Windows-based environment called "Elegance" (Spronck & Kerckhoffs, 1997), that is particularly suited to experiment with different configurations for the evolutionary algorithm.

The Khepera robot possesses eight infra-red light and proximity sensors (Figure 2). In our experiments, we disregarded the light sensors and only used the proximity values (except for the experiments for determining a suitable fitness function). The sensors are numbered clockwise from 0 to 7 starting from the left of the robot. Sensors 2 and 3 point forward, while sensors 6 and 7 point backwards. To control the robot, the two wheels of the robot are supplied with input values

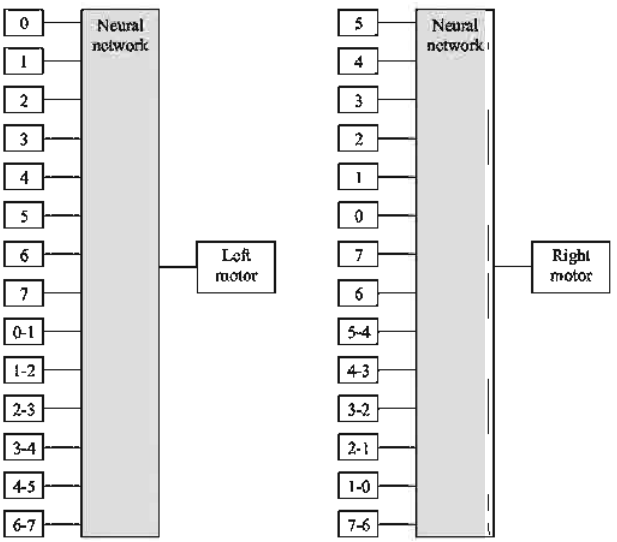
ranging from –10 to +10 in integer numbers, where the sign of the input value determines the direction of rotation of the wheels. The robot moves in discrete steps and requires new inputs at each step.

As can be seen from Figure 1, the robot is placed within a short distance from the box, close to the back wall. In order to move the box as far as possible from its initial position, the only viable pushing direction is towards the top of the area. The two walls are rough. The robot may use a wall as support while pushing the box, but then has to deal with the roughness that may cause the box to get stuck.

The Controller

The neural controller we use has 14 inputs. Eight inputs are delivered by the proximity sensors. The other six are defined as the differences between the values of neighbouring proximity sensors (leaving out the differences between sensors 5 and 6 and between sensors 0 and 7, because these pairs are too widely separated). We call these (virtual) sensors "edge detectors," because they deliver large inputs when an edge (i.e., a spatial discontinuity in proximity values) is detected. In a mathematical sense, the virtual sensors are redundant. However, in our earlier

Figure 3: Exploiting the mirror symmetry of the robot to derive a neural controller for one wheel from the neural controller for the other wheel. The left network drives the left motor, the right network the right motor. The network inputs are proximity values derived from the Khepera robot shown in Figure 2. The neural networks are equal, but the inputs have been moved around and the signs of the edge-detecting inputs have been switched.



studies we found them to be beneficial to the evolution process (Sprinkhuizen-Kuyper, 2001). The use of edge-detecting inputs is inspired by biological visual systems which are more sensitive to differences than to the absolute level of light intensity. For instance, in the human visual system, edge-detecting neurons are omni-present (Cornsweet, 1970). Detecting edges is an important prerequisite for visual-guided behaviour, and allows the controller to distinguish the box from the walls.

Since the robot has two wheels, either a neural network with two outputs is needed, or two separate neural networks are needed. Because of the symmetric placement of the sensors around the robot (see Figure 2), a mirrored copy of a neural network that drives one of the wheels can be used to drive the other wheel. The mirrored copy of the network requires exchanged inputs for the proximity and virtual sensors. In addition, the signals delivered by the virtual edge-detecting sensors have to be negated (see Figure 3). The use of two (almost) identical networks reduces the number of free parameters considerably which makes the search for a solution easier. The output of the neural networks is mapped onto a legal interval for the motor values by using a sigmoid transfer function.

The neural activation functions employed in our neural controller are defined as linear functions. Although the use of linear transfer functions in a multi-layer network does not make much sense from a mathematical viewpoint (a multi-layer network of linear layers can always be reduced to a single linear-layer network), preliminary results revealed only small differences in performance with the (traditional) non-linear transfer functions (Sprinkhuizen-Kuyper, 2001). It turns out that the extra layer may help the evolution process by keeping the connection weights relatively small.

The Evolutionary Algorithm

The evolutionary algorithm used in our experiments works on a population of about 100 individuals. Tournament selection with size 3 is used to get parents for the genetic operators. Newly generated individuals are inserted back in the original population, using a crowding scheme with a factor of 3. We used elitism to prevent loss of the best solutions, and the evolution process continues until no significant increase in fitness is visible any more. Usually this takes 25 to 35 generations.

The chromosome representing a neural network consists of an array of "connection genes." Each connection gene represents a single possible connection of the network and is defined as a pair of one bit and one real number. The bit represents the presence or absence of a connection and the real value specifies the weight of the connection. During absence, the weight value is maintained in the chromosome which facilitates the evolution process by functioning as a "memory" for removed weight values.

Computational Intelligence in Control

Masoud Mohammadian
Ruhul Amin Sarker
Xin Yao

IDEA GROUP PUBLISHING

Computational Intelligence in Control

Masoud Mohammadian, University of Canberra, Australia
Ruhul Amin Sarker, University of New South Wales, Australia
Xin Yao, University of Birmingham, UK

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore • Beijing

Acquisition Editor:	Mehdi Khosrowpour
Senior Managing Editor:	Jan Travers
Managing Editor:	Amanda Appicello
Development Editor:	Michele Rossi
Copy Editor:	Maria Boyer
Typesetter:	Tamara Gillis
Cover Design:	Integrated Book Technology
Printed at:	Integrated Book Technology

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Mohammadian, Masoud.
Computational intelligence in control / Masoud Mohammadian, Ruhul Amin Sarker and Xin Yao.
p. cm.
ISBN 1-59140-037-6 (hardcover) -- ISBN 1-59140-079-1 (ebook)
1. Neural networks (Computer science) 2. Automatic control. 3. Computational intelligence. I. Amin, Ruhul. II. Yao, Xin, 1962- III. Title.
QA76.87 .M58 2003
006.3--dc21

2002014188

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Computational Intelligence in Control

Table of Contents

Preface

This book covers the recent applications of computational intelligence techniques for modelling, control and automation. The application of these techniques has been found useful in problems when the process is either difficult to model or difficult to solve by conventional methods. There are numerous practical applications of computational intelligence techniques in modelling, control, automation, prediction, image processing and data mining.

Research and development work in the area of computational intelligence is growing rapidly due to the many successful applications of these new techniques in very diverse problems. "Computational Intelligence" covers many fields such as neural networks, (adaptive) fuzzy logic, evolutionary computing, and their hybrids and derivatives. Many industries have benefited from adopting this technology. The increased number of patents and diverse range of products developed using computational intelligence methods is evidence of this fact.

These techniques have attracted increasing attention in recent years for solving many complex problems. They are inspired by nature, biology, statistical techniques, physics and neuroscience. They have been successfully applied in solving many complex problems where traditional problem-solving methods have failed. These modern techniques are taking firm steps as robust problem-solving mechanisms.

This volume aims to be a repository for the current and cutting-edge applications of computational intelligent techniques in modelling control and automation, an area with great demand in the market nowadays.

With roots in modelling, automation, identification and control, computational intelligence techniques provide an interdisciplinary area that is concerned with learning and adaptation of solutions for complex problems. This instantiated an enormous amount of research, searching for learning methods that are capable of controlling novel and non-trivial systems in different industries.

This book consists of open-solicited and invited papers written by leading researchers in the field of computational intelligence. All full papers have been peer review by at least two recognised reviewers. Our goal is to provide a book

that covers the foundation as well as the practical side of the computational intelligence.

The book consists of 17 chapters in the fields of self-learning and adaptive control, robotics and manufacturing, machine learning, evolutionary optimisation, information retrieval, fuzzy logic, Bayesian systems, neural networks and hybrid evolutionary computing.

This book will be highly useful to postgraduate students, researchers, doctoral students, instructors, and partitioners of computational intelligence techniques, industrial engineers, computer scientists and mathematicians with interest in modelling and control.

We would like to thank the senior and assistant editors of Idea Group Publishing for their professional and technical assistance during the preparation of this book. We are grateful to the unknown reviewers for the book proposal for their review and approval of the book proposal. Our special thanks goes to Michele Rossi and Mehdi Khosrowpour for their assistance and their valuable advise in finalizing this book.

We would like to acknowledge the assistance of all involved in the collation and review process of the book, without whose support and encouragement this book could not have been successfully completed.

We wish to thank all the authors for their insights and excellent contributions to this book. We would like also to thank our families for their understanding and support throughout this book project.

M. Mohammadian, R. Sarker and X. Yao

SECTION I:

NEURAL NETWORKS DESIGN, CONTROL AND ROBOTICS APPLICATION

Chapter I

Designing Neural Network Ensembles by Minimising Mutual Information

Yong Liu
The University of Aizu, Japan

Xin Yao
The University of Birmingham, UK

Tetsuya Higuchi
National Institute of Advanced Industrial Science and Technology, Japan

ABSTRACT

This chapter describes negative correlation learning for designing neural network ensembles. Negative correlation learning has been firstly analysed in terms of minimising mutual information on a regression task. By minimising the mutual information between variables extracted by two neural networks, they are forced to convey different information about some features of their input. Based on the decision boundaries and correct response sets, negative correlation learning has been further studied on two pattern classification problems. The purpose of examining the decision boundaries and the correct response sets is not only to illustrate the learning behavior of negative correlation learning, but also to cast light on how to design more effective neural network ensembles. The experimental results showed the decision boundary of the trained neural network ensemble by negative correlation learning is almost as good as the optimum decision boundary.

INTRODUCTION

In single neural network methods, the neural network learning problem is often formulated as an optimisation problem, i.e., minimising certain criteria, e.g., minimum error, fastest learning, lowest complexity, etc., about architectures. Learning algorithms, such as backpropagation (BP) (Rumelhart, Hinton & Williams, 1986), are used as optimisation algorithms to minimise an error function. Despite the different error functions used, these learning algorithms reduce a learning problem to the same kind of optimisation problem.

Learning is different from optimisation because we want the learned system to have best generalisation, which is different from minimising an error function. The neural network with the minimum error on the training set does not necessarily have the best generalisation unless there is an equivalence between generalisation and the error function. Unfortunately, measuring generalisation exactly and accurately is almost impossible in practice (Wolpert, 1990), although there are many theories and criteria on generalisation, such as the minimum description length (Rissanen, 1978), Akaike's information criteria (Akaike, 1974) and minimum message length (Wallace & Patrick, 1991). In practice, these criteria are often used to define better error functions in the hope that minimising the functions will maximise generalisation. While better error functions often lead to better generalisation of learned systems, there is no guarantee. Regardless of the error functions used, single network methods are still used as optimisation algorithms. They just optimise different error functions. The nature of the problem is unchanged.

While there is little we can do in single neural network methods, there are opportunities in neural network ensemble methods. Neural network ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, a neural network ensemble combines a set of neural networks which learn to subdivide the task and thereby solve it more efficiently and elegantly. A neural network ensemble offers several advantages over a monolithic neural network. First, it can perform more complex tasks than any of its components (i.e., individual neural networks in the ensemble). Secondly, it can make an overall system easier to understand and modify. Finally, it is more robust than a monolithic neural network and can show graceful performance degradation in situations where only a subset of neural networks in the ensemble are performing correctly. Given the advantages of neural network ensembles and the complexity of the problems that are beginning to be investigated, it is clear that the neural network ensemble method will be an important and pervasive problem-solving technique.

The idea of designing an ensemble learning system consisting of many subsystems can be traced back to as early as 1958 (Selfridge, 1958; Nilsson, 1965). Since the early 1990s, algorithms based on similar ideas have been developed in many different but related forms, such as neural network ensembles

(Hansen & Salamon, 1990; Sharkey, 1996), mixtures of experts (Jacobs, Jordan, Nowlan & Hinton, 1991; Jacobs & Jordan, 1991; Jacobs, Jordan & Barto, 1991; Jacobs, 1997), various boosting and bagging methods (Drucker, Cortes, Jackel, LeCun & Vapnik, 1994; Schapire, 1990; Drucker, Schapire & Simard, 1993) and many others. There are a number of methods of designing neural network ensembles. To summarise, there are three ways of designing neural network ensembles in these methods: independent training, sequential training and simultaneous training.

A number of methods have been proposed to train a set of neural networks independently by varying initial random weights, the architectures, the learning algorithm used and the data (Hansen et al., 1990; Sarkar, 1996). Experimental results have shown that networks obtained from a given network architecture for different initial random weights often correctly recognize different subsets of a given test set (Hansen et al., 1990; Sarkar, 1996). As argued in Hansen et al. (1990), because each network makes generalisation errors on different subsets of the input space, the collective decision produced by the ensemble is less likely to be in error than the decision made by any of the individual networks.

Most independent training methods emphasised independence among individual neural networks in an ensemble. One of the disadvantages of such a method is the loss of interaction among the individual networks during learning. There is no consideration of whether what one individual learns has already been learned by other individuals. The errors of independently trained neural networks may still be positively correlated. It has been found that the combining results are weakened if the errors of individual networks are positively correlated (Clemen & Winkler, 1985). In order to decorrelate the individual neural networks, sequential training methods train a set of networks in a particular order (Drucker et al., 1993; Opitz & Shavlik, 1996; Rosen, 1996). Drucker et al. (1993) suggested training the neural networks using the boosting algorithm. The boosting algorithm was originally proposed by Schapire (1990). Schapire proved that it is theoretically possible to convert a weak learning algorithm that performs only slightly better than random guessing into one that achieves arbitrary accuracy. The proof presented by Schapire (1990) is constructive. The construction uses filtering to modify the distribution of examples in such a way as to force the weak learning algorithm to focus on the harder-to-learn parts of the distribution.

Most of the independent training methods and sequential training methods follow a two-stage design process: first generating individual networks, and then combining them. The possible interactions among the individual networks cannot be exploited until the integration stage. There is no feedback from the integration stage to the individual network design stage. It is possible that some of the independently designed networks do not make much contribution to the integrated system. In

order to use the feedback from the integration, simultaneous training methods train a set of networks together. Negative correlation learning (Liu & Yao, 1998a, 1998b, 1999) and the mixtures-of-experts (ME) architectures (Jacobs et al., 1991; Jordan & Jacobs, 1994) are two examples of simultaneous training methods. The idea of negative correlation learning is to encourage different individual networks in the ensemble to learn different parts or aspects of the training data, so that the ensemble can better learn the entire training data. In negative correlation learning, the individual networks are trained simultaneously rather than independently or sequentially. This provides an opportunity for the individual networks to interact with each other and to specialise.

In this chapter, negative correlation learning has been firstly analysed in terms of minimising mutual information on a regression task. The similarity measurement between two neural networks in an ensemble can be defined by the explicit mutual information of output variables extracted by two neural networks. The mutual information between two variables, output F_i of network i and output F_j of network j, is given by

$$I(F_i; F_j) = h(F_i) + h(F_j) - h(F_i, F_j) \tag{1}$$

where $h(F_i)$ is the entropy of F_i, $h(F_j)$ is the entropy of F_j, and $h(F_i, F_j)$ is the joint differential entropy of F_i and F_j. The equation shows that joint differential entropy can only have high entropy if the mutual information between two variables is low, while each variable has high individual entropy. That is, the lower mutual information two variables have, the more different they are. By minimising the mutual information between variables extracted by two neural networks, they are forced to convey different information about some features of their input. The idea of minimising mutual information is to encourage different individual networks to learn different parts or aspects of the training data so that the ensemble can learn the whole training data better.

Based on the decision boundaries and correct response sets, negative correlation learning has been further studied on two pattern classification problems. The purpose of examining the decision boundaries and the correct response sets is not only to illustrates the learning behavior of negative correlation learning, but also to cast light on how to design more effective neural network ensembles. The experimental results showed the decision boundary of the trained neural network ensemble by negative correlation learning is almost as good as the optimum decision boundary.

The rest of this chapter is organised as follows: Next, the chapter explores the connections between the mutual information and the correlation coefficient, and

explains how negative correlation learning can be used to minimise mutual information; then the chapter analyses negative correlation learning via the metrics of mutual information on a regression task; the chapter then discusses the decision boundaries constructed by negative correlation learning on a pattern classification problem; finally the chapter examines the correct response sets of individual networks trained by negative correlation learning and their intersections, and the chapter concludes with a summary of the chapter and a few remarks.

MINIMISING MUTUAL INFORMATION BY NEGATIVE CORRELATION LEARNING

Minimisation of Mutual Information

Suppose the output F_i of network i and the output F_j of network j are Gaussian random variables. Their variances are σ_i^2 and σ_j^2, respectively. The mutual information between F_i and F_j can be defined by Eq.(1) (van der Lubbe, 1997, 1999). The differential entropy $h(F_i)$ and $h(F_j)$ are given by

$$h(F_i) = [1 + log(2\pi\sigma_i^2)] / 2 \tag{2}$$

and

$$h(F_j) = [1 + log(2\pi\sigma_j^2)] / 2 \tag{3}$$

The joint differential entropy $h(F_i, F_j)$ is given by

$$h(F_i, F_j) = 1 + log(2\pi) + log|det(\Sigma)| \tag{4}$$

where Σ is the 2-by-2 covariance matrix of F_i and F_j. The determinant of Σ is

$$det(\Sigma) = \sigma_i^2\sigma_j^2(1 - \rho_{ij}^2) \tag{5}$$

where ρ_{ij} is the correlation coefficient of F_i and F_j

$$\rho_{ij} = E[(F_i - E[F_i])(F_j - E[F_j])] / (\sigma_i^2 \sigma_j^2) \quad (6)$$

Using the formula of Eq.(5), we get

$$h(F_i, F_j) = 1 + log(2\pi) + log[\sigma_i^2 \sigma_j^2 (1 - \rho_{ij}^2)] / 2 \quad (7)$$

By substituting Eqs.(2), (3), and (7) in (1), we get

$$I(F_i ; F_j) = - log(1 - \rho_{ij}^2) / 2 \quad (8)$$

From Eq.(8), we may make the following statements:

1. If F_i and F_j are uncorrelated, the correlation coefficient ρ_{ij} is reduced to zero, and the mutual information $I(F_i ; F_j)$ becomes very small.
2. If F_i and F_j are highly positively correlated, the correlation coefficient ρ_{ij} is close to 1, and mutual information $I(F_i ; F_j)$ becomes very large.

Both theoretical and experimental results (Clemen et al., 1985) have indicated that when individual networks in an ensemble are unbiased, average procedures are most effective in combining them when errors in the individual networks are negatively correlated and moderately effective when the errors are uncorrelated. There is little to be gained from average procedures when the errors are positively correlated. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of the population should be minimised. Minimising the mutual information between each individual neural network and the rest of the population is equivalent to minimising the correlation coefficient between them.

Negative Correlation Learning

Given the training data set $D = \{(\mathbf{x}(1),\mathbf{y}(1)), \ldots, (\mathbf{x}(N),\mathbf{y}(N))\}$, we consider estimating $\mathbf{y}$ by forming a neural network ensemble whose output is a simple averaging of outputs F_i of a set of neural networks. All the individual networks in the ensemble are trained on the same training data set D

$$F(n) = \frac{1}{M} \sum_{i=1}^{M} F_i(n) \quad (9)$$

where $F_i(n)$ is the output of individual network i on the nth training pattern $\mathbf{x}(n)$, $F(n)$ is the output of the neural network ensemble on the nth training pattern, and M is the number of individual networks in the neural network ensemble.

The idea of negative correlation learning is to introduce a correlation penalty term into the error function of each individual network so that the individual network can be trained simultaneously and interactively. The error function E_i for individual i on the training data set D in negative correlation learning is defined by

$$E_i = \frac{1}{N} \sum_{n=1}^{N} \left[\frac{1}{2} (F_i(n) - y(n))^2 + \lambda p_i(n) \right] \tag{10}$$

where N is the number of training patterns, $E_i(n)$ is the value of the error function of network i at presentation of the nth training pattern and $y(n)$ is the desired output of the nth training pattern. The first term in the right side of Eq.(10) is the mean-squared error of individual network i. The second term p_i is a correlation penalty function. The purpose of minimising p_i is to negatively correlate each individual's error with errors for the rest of the ensemble. The parameter λ is used to adjust the strength of the penalty.

The penalty function p_i has the form

$$p_i(n) = -(F_i(n)-F(n))^2 / 2 \tag{11}$$

The partial derivative of E_i with respect to the output of individual i on the nth training pattern is

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F_i(n) - y(n) - \lambda(F_i(n) - F(n)) \tag{12}$$

where we have made use of the assumption that the output of ensemble $F(n)$ has constant value with respect to $F_i(n)$. The value of parameter λ lies inside the range $0 \le \lambda \le 1$ so that both $(1-\lambda)$ and λ have nonnegative values. BP (Rumelhart et al., 1996) algorithm has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eq.(12) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an *epoch*. Negative correlation learning from Eq.(12) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda(F_i(n) - F(n))$ for the ith neural network.

From Eqs.(10), (11) and (12), we may make the following observations:

1. During the training process, all the individual networks interact with each other through their penalty terms in the error functions. Each network F_i minimises not only the difference between $F_i(n)$ and $y(n)$, but also the difference between $F(n)$ and $y(n)$. That is, negative correlation learning considers errors what all other neural networks have learned while training a neural network.
2. For $\lambda=0.0$, there are no correlation penalty terms in the error functions of the individual networks, and the individual networks are just trained independently using BP. That is, independent training using BP for the individual networks is a special case of negative correlation learning.
3. For $\lambda=1$, from Eq.(12) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - y(n) \tag{13}$$

Note that the error of the ensemble for the nth training pattern is defined by

$$E_{ensemble} = \frac{1}{2}\left(\frac{1}{M}\sum_{i=1}^{M} F_i(n) - y(n)\right)^2 \tag{14}$$

The partial derivative of $E_{ensemble}$ with respect to F_i on the nth training pattern is

$$\frac{\partial E_{ensemble}}{\partial F_i(n)} = \frac{1}{M}(F(n) - y(n)) \tag{15}$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ensemble}}{\partial F_i(n)} \tag{16}$$

The minimisation of the error function of the ensemble is achieved by minimising the error functions of the individual networks. From this point of view, negative correlation learning provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual networks.

ANALYSIS BASED ON MEASURING MUTUAL INFORMATION

In order to understand why and how negative correlation learning works, this section analyses it through measuring mutual information on a regression task in three cases: noise-free condition, small noise condition and large noise condition.

Simulation Setup

The regression function investigated here is

$$f(x) = \frac{1}{13}[10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5)] - 1 \quad (17)$$

where $\mathbf{x}=[x_1, ..., x_5]$ is an input vector whose components lie between zero and one. The value of $f(\mathbf{x})$ lies in the interval [-1, 1]. This regression task has been used by Jacobs (1997) to estimate the bias of mixture-of-experts architectures and the variance and covariance of experts' weighted outputs.

Twenty-five training sets, $(x^{(k)}(l), y^{(k)}(l))$, $l=1, ..., L$, $L=500$, $k=1, ..., K$, $K=25$, were created at random. Each set consisted of 500 input-output patterns in which the components of the input vectors were independently sampled from a uniform distribution over the interval (0, 1). In the noise-free condition, the target outputs were not corrupted by noise; in the small noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2=0.1$ to the function $f(\mathbf{x})$; in the large noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2=0.2$ to the function $f(\mathbf{x})$. A testing set of 1,024 input-output patterns, $(\mathbf{t}(n), d(n))$, $n = 1, ..., N$, $N = 1024$, was also generated. For this set, the components of the input vectors were independently sampled from a uniform distribution over the interval (0, 1), and the target outputs were not corrupted by noise in all three conditions. Each individual network in the ensemble is a multi-layer perceptron with one hidden layer. All the individual networks have 5 hidden nodes in an ensemble architecture. The hidden node function is defined by the logistic function

$$\varphi(y) = \frac{1}{1+\exp(-y)} \quad (18)$$

The network output is a linear combination of the outputs of the hidden nodes.

For each estimation of mutual information among an ensemble, 25 simulations were conducted. In each simulation, the ensemble was trained on a different training set from the same initial weights distributed inside a small range so that different simulations of an ensemble yielded different performances solely due to the use of different training sets. Such simulation setup follows the suggestions from Jacobs (1997).

Measurement of Mutual Information

The average outputs of the ensemble and the individual network i on the nth pattern in the testing set, (t(n), d(n)), n = 1, ..., N, are denoted and given respectively by

$$\overline{F}(\mathrm{t}(n)) = \frac{1}{K}\sum_{k=1}^{K} F^{(k)}(\mathrm{t}(n)) \tag{19}$$

and

$$\overline{F}_i(\mathrm{t}(n)) = \frac{1}{K}\sum_{k=1}^{K} F_i^{(k)}(\mathrm{t}(n)) \tag{20}$$

where $F^{(k)}(t(n))$ and $F_i^{(k)}(t(n))$ are the outputs of the ensemble and the individual network i on the nth pattern in the testing set from the kth simulation, respectively, and $K=25$ is the number of simulations. From Eq.(6), the correlation coefficient between network i and network j is given by

$$\rho_{ij} = \frac{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_i^{(k)}(\mathrm{t}(n)) - \overline{F}_i(\mathrm{t}(n))\right)\left(F_j^{(k)}(\mathrm{t}(n)) - \overline{F}_j(\mathrm{t}(n))\right)}{\sqrt{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_i^{(k)}(\mathrm{t}(n)) - \overline{F}_i(\mathrm{t}(n))\right)^2 \sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_j^{(k)}(\mathrm{t}(n)) - \overline{F}_j(\mathrm{t}(n))\right)^2}} \tag{21}$$

From Eq.(8), the integrated mutual information among the ensembles can be defined by

$$E_{mi} = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1, j\neq i}^{M} log(1-\rho_{ij}^2) \tag{22}$$

We may also define the integrated mean-squared error (MSE) on the testing set as

$$E_{mse} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{K}\sum_{k=1}^{K}(F^{(k)}(\mathrm{t}(n)) - d(n))^2 \quad (23)$$

The integrated mean-squared error E_{train_mse} on the training set is given by

$$E_{train\_mis} = \frac{1}{L}\sum_{l=1}^{L}\frac{1}{K}\sum_{k=1}^{K}(F^{(k)}(\mathrm{x}^{(k)}(l)) - y^{(k)}(l))^2 \quad (24)$$

Results in the Noise-Free Condition

The results of negative correlation learning in the noise-free condition for the different values of λ at epoch 2000 are given in Table 1. The results suggest that both E_{train_mse} and E_{test_mse} appeared to decrease with the increasing value of λ. The mutual information E_{mi} among the ensemble decreased as the value of λ increased when $0 \le \lambda \le 0.5$. However, when λ increased further to 0.75 and 1, the mutual information E_{mi} had larger values. The reason of having larger mutual information at $\lambda = 0.75$ and $\lambda = 1$ is that some correlation coefficients had negative values and the mutual information depends on the absolute values of correlation coefficients.

In order to find out why E_{train_mse} decreased with increasing value of λ, the concept of capability of a trained ensemble is introduced. The capability of a trained ensemble is measured by its ability of producing correct input-output mapping on the training set used, specifically, by its integrated mean-squared error E_{train_mse} on the training set. The smaller E_{train_mse} is, the larger capability the trained ensemble has.

Results in the Noise Conditions

Table 2 and Table 3 compare the performance of negative correlation learning for different strength parameters in both small noise (variance $\sigma^2 = 0.1$) and large

Table 1: The results of negative correlation learning in the noise-free condition for different l values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	0.3706	0.1478	0.1038	0.1704	0.6308
$E_{test\ mse}$	0.0016	0.0013	0.0011	0.0007	0.0002
$E_{train\ mse}$	0.0013	0.0010	0.0008	0.0005	0.0001

noise (variance $\sigma^2 = 0.2$) conditions. The results show that there were same trends for E_{mi}, E_{test_mse} and E_{train_mse} in both noise-free and noise conditions when $\lambda \leq 0.5$. That is, E_{mi}, E_{test_mse} and E_{train_mse} appeared to decrease with the increasing value of λ. However, E_{test_mse} appeared to decrease first and then increase with the increasing value of λ.

In order to find out why E_{test_mse} showed different trends in noise-free and noise conditions when $\lambda = 0.75$ and $\lambda = 1$, the integrated mean-squared error E_{train_mse} on the training set was also shown in Tables 1, 2 and 3. When $\lambda = 0$, the neural network ensemble trained had relatively large E_{train_mse}. It indicated that the capability of the neural network ensemble trained was not big enough to produce correct input-output mapping (i.e., it was underfitting) for this regression task. When $\lambda = 1$, the neural network ensemble trained learned too many specific input-output relations (i.e., it was overfitting), and it might memorise the training data and therefore be less able to generalise between similar input-output patterns. Although the overfitting was not observed for the neural network ensemble used in the noise-free condition, too large capability of the neural network ensemble will lead to overfitting for both noise-free and noise conditions because of the ill-posedness of any finite training set (Friedman, 1994).

Choosing a proper value of λ is important, and also problem dependent. For the noise conditions used for this regression task and the ensemble architectured used, the performance of the ensemble was optimal for $\lambda = 0.5$ among the tested values of λ in the sense of minimising the MSE on the testing set.

Table 2: The results of negative correlation learning in the small noise condition for different λ values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	6.5495	3.8761	1.4547	0.3877	0.2431
E_{test_mse}	0.0137	0.0128	0.0124	0.0126	0.0290
E_{train_mse}	0.0962	0.0940	0.0915	0.0873	0.0778

Table 3: The results of negative correlation learning in the large noise condition for different λ values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	6.7503	3.9652	1.6957	0.4341	0.2030
E_{test_mse}	0.0249	0.0235	0.0228	0.0248	0.0633
E_{train_mse}	0.1895	0.1863	0.1813	0.1721	0.1512

ANALYSIS BASED ON DECISION BOUNDARIES

This section analyses the decision boundaries constructed by both negative correlation learning and the independent training. The independent training is a special case of negative correlation learning for $\lambda = 0.0$ in Eq.(12).

Simulation Setup

The objective of the pattern classification problem is to distinguish between two classes of overlapping, two-dimensional, Gaussian-distributed patterns labeled 1 and 2. Let Class 1 and Class 2 denote the set of events for which a random vector **x** belongs to patters 1 and 2, respectively. We may then express the conditional probability density functions for the two classes:

$$f_X(x) = \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{1}{2\sigma_1^2}||x - \mu_1||^2\right) \tag{25}$$

where mean vector $\mu_1 = [0,0]^T$ and variance $\sigma_1^2 = 1$.

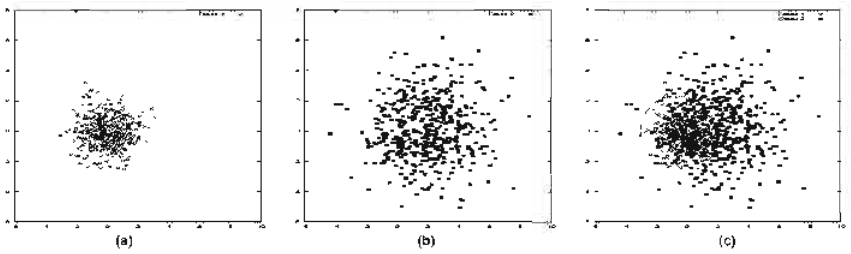
$$f_X(x) = \frac{1}{2\pi\sigma_2^2} \exp\left(-\frac{1}{2\sigma_2^2}||x - \mu_2||^2\right) \tag{26}$$

where mean vector $\mu_2 = [0,0]^T$ and variance $\sigma_2^2 = 4$. The two classes are assumed to be equiprobable; that is $p_1 = p_2 = ½$. The costs for misclassifications are assumed to be equal, and the costs for correct classifications are assumed to be zero. On this basis, the (optimum) Bayes classifier achieves a probability of correct classification $p_c = 81.51$ percent. The boundary of the Bayes classifier consists of a circle of center $[-2/3, 0]^T$ and radius $r = 2.34$; 1000 points from each of two processes were generated for the training set. The testing set consists of 16,000 points from each of two classes.

Figure 1 shows individual scatter diagrams for classes and the joint scatter diagram representing the superposition of scatter plots of 500 points from each of two processes. This latter diagram clearly shows that the two distributions overlap each other significantly, indicating that there is inevitably a significant probability of misclassification.

The ensemble architecture used in the experiments has three networks. Each individual network in the ensemble is a multi-layer perceptron with one hidden layer. All the individual networks have three hidden nodes in an ensemble architecture.

Figure 1: (a) Scatter plot of Class 1; (b) Scatter plot of Class 2; (c) Combined scatter plot of both classes; the circle represents the optimum Bayes solution



Both hidden node function and output node function are defined by the logistic function in Eq.(18).

Experimental Results

The results presented in Table 4 pertain to 10 different runs of the experiment, with each run involving the use of 2,000 data points for training and 32,000 for testing. Figures 2 and 3 compare the decision boundaries constructed by negative

Figure 2: Decision boundaries formed by the different networks trained by the negative correlation learning ($\lambda = 0.75$): (a) Network 1; (b) Network 2; (c) Network 3; (d) Ensemble; the circle represents the optimum Bayes solution

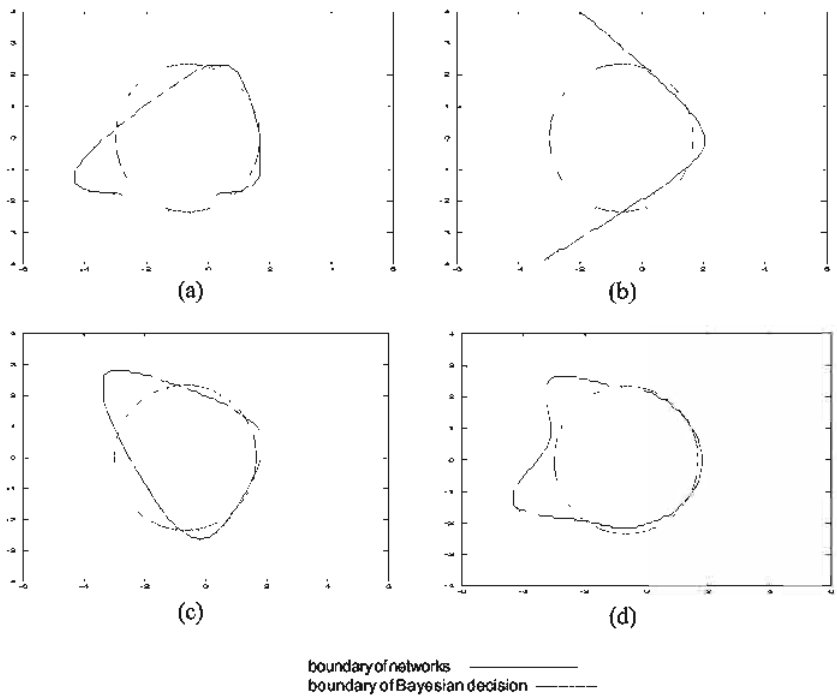
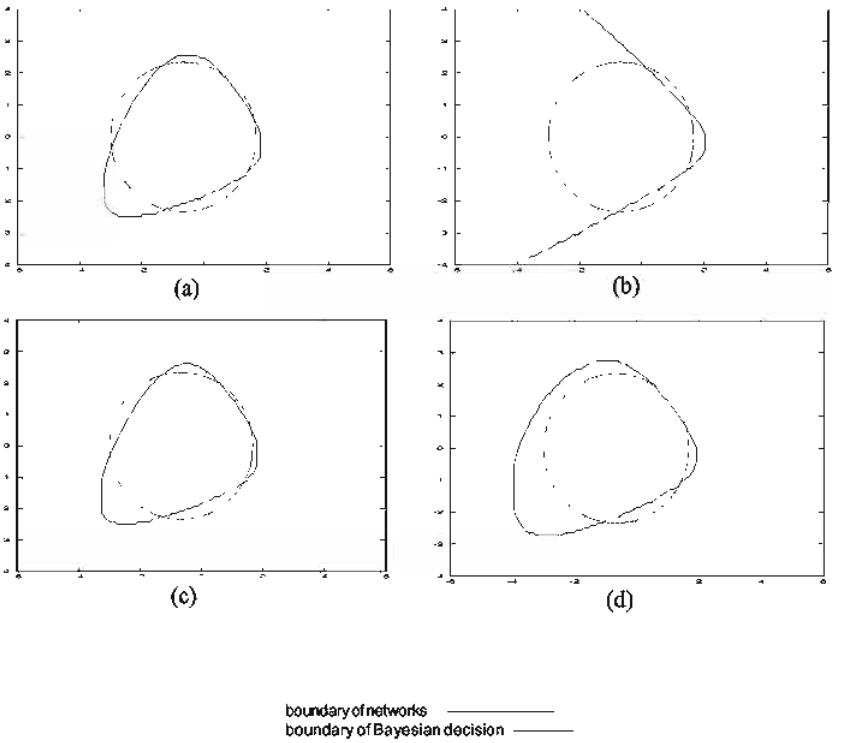


Figure 3: Decision boundaries formed by the different networks trained by the independent training (i.e., $\lambda = 0.0$ in negative correlation learning): (a) Network 1; (b) Network 2; (c) Network 3; (d) Ensemble; the circle represents the optimum Bayes solution



correlation learning and the independent training. In comparison of the average correct classification percentage and the decision boundaries obtained by the two ensemble learning methods, it is clear that negative correlation learning outperformed the independent training method. Although the classification performance of individual networks in the independent training is relatively good, the overall performance of the entire ensemble was not improved because different networks, such as Network 1 and Network 3 in Figure 3, tended to generate the similar decision boundaries.

The percentage of correct classification of the ensemble trained by negative correlation is 81.41, which is almost equal to that realised by the Bayesian classifier. Figure 2 clearly demonstrates that negative correlation learning is capable of constructing a decision between Class 1 and Class 2 that is almost as good as the optimum decision boundary. It is evident from Figure 2 that different individual networks trained by negative correlation learning were able to specialise to different parts of the testing set.

Table 4: Comparison between negativecorrelation learning (NCL) ($\lambda = 0.75$) and the independent training (i.e., $\lambda = 0.0$ in negative correlation learning) on the classification performance of individual networks and the ensemble; the results are the average correct classification percentage on the testing set over 10 independent runs

Methods	Net 1	Net 2	Net 3	Ensemble
NCL	81.11	75.26	73.09	81.03
Independent Training	81.13	80.49	81.13	80.99

ANALYSIS BASED ON THE CORRECT RESPONSE SETS

In this section, negative correlation learning was tested on the Australian credit card assessment problem. The problem is how to assess applications for credit cards based on a number of attributes. There are 690 patterns in total. The output has two classes. The 14 attributes include 6 numeric values and 8 discrete ones, the latter having from 2 to 14 possible values. The Australian credit card assessment problem is a classification problem which is different from the regression type of tasks, whose outputs are continuous. The data set was obtained from the UCI machine learning benchmark repository. It is available by anonymous ftp at ics.uci.edu (128.195.1.1) in directory /pub/machine-learning-databases.

Experimental Setup

The data set was partitioned into two sets: a training set and a testing set. The first 518 examples were used for the training set, and the remaining 172 examples for the testing set. The input attributes were rescaled to between 0.0 and 1.0 by a linear function. The output attributes of all the problems were encoded using a *1-of-m* output representation for m classes. The output with the highest activation designated the class. The aim of this section is to study the difference between negative correlation learning and independent training, rather than to compare negative correlation learning with previous work. The experiments used such a single train-and-test partition.

The ensemble architecture used in the experiments has 4 networks. Each individual network is a feedforward network with one hidden layer. Both hidden node function and output node function are defined by the logistic function in Eq.(18). All the individual networks have 10 hidden nodes. The number of training

epochs was set to 250. The strength parameter λ was set to 1.0. These parameters were chosen after limited preliminary experiments. They are not meant to be optimal.

Experimental Results

Table 5 shows the average results of negative correlation learning over 25 runs. Each run of negative correlation learning was from different initial weights. The ensemble with the same initial weight setup was also trained using BP without the correlation penalty terms (i.e., $\lambda = 0.0$ in negative correlation learning). Results are also shown in Table 5. For this problem, the simple averaging defined in Eq.(9) was first applied to decide the output of the ensemble. For the simple averaging, it was surprising that the results of negative correlation learning with $\lambda = 1.0$ were similar to those of independent training. This phenomenon seems contradictory to the claim that the effect of the correlation penalty term is to encourage different individual networks in an ensemble to learn different parts or aspects of the training data. In order to verify and quantify this claim, we compared the outputs of the individual networks trained with the correlation penalty terms to those of the individual networks trained without the correlation penalty terms.

Table 5: Comparison of error rates between negative correlation learning ($\lambda = 1.0$) and independent training (i.e., $\lambda = 0.0$ in negative correlation learning) on the Australian credit card assessment problem; the results were averaged over 25 runs. "Simple Averaging" and "Winner-Takes-All" indicate two different combination methods used in negative correlation learning, Mean, SD, Min and Max indicate the mean value,standard deviation, minimum and maximum value, respectively

	Error Rate	Simple Averageing	Winner-Takes-All
$\lambda = 1.0$	Mean	0.1337	0.1195
	SD	0.0068	0.0052
	Min	0.1163	0.1105
	Max	0.1454	0.1279
$\lambda = 0.0$	Mean	0.1368	0.1384
	SD	0.0048	0.0049
	Min	0.1279	0.1279
	Max	0.1454	0.1512

Table 6: The sizes of the correct response sets of individual networks created respectively by negative correlation learning (λ = 1.0) and independent training (i.e., λ = 0.0 in negative correlation learning) on the testing set and the sizes of their intersections for the Australian credit card assessment problem; the results were obtained from the first run among the 25 runs

$\lambda = 1.0$			$\lambda = 0.0$		
$\Omega_1 = 147$	$\Omega_2 = 143$	$\Omega_2 = 138$	$\Omega_1 = 149$	$\Omega_2 = 147$	$\Omega_2 = 148$
$\Omega_4 = 143$	$\Omega_{12} = 138$	$\Omega_{13} = 124$	$\Omega_4 = 148$	$\Omega_{12} = 147$	$\Omega_{13} = 147$
$\Omega_{14} = 141$	$\Omega_{23} = 116$	$\Omega_{24} = 133$	$\Omega_{14} = 147$	$\Omega_{23} = 147$	$\Omega_{24} = 146$
$\Omega_{34} = 123$	$\Omega_{123} = 115$	$\Omega_{124} = 133$	$\Omega_{34} = 146$	$\Omega_{123} = 147$	$\Omega_{124} = 146$
$\Omega_{134} = 121$	$\Omega_{234} = 113$	$\Omega_{1234} = 113$	$\Omega_{134} = 146$	$\Omega_{234} = 146$	$\Omega_{1234} = 146$

Two notions were introduced to analyse negative correlation learning. They are the correct response sets of individual networks and their intersections. The correct response set S_i of individual network *i* on the testing set consists of all the patterns in the testing set which are classified correctly by the individual network *i*. Let Ω_i denote the size of set S_i, and $\Omega_{i1i2\cdots ik}$ denote the size of set $S_{i1} \cap S_{i2} \cap \cdots \cap S_{ik}$. Table 6 shows the sizes of the correct response sets of individual networks and their intersections on the testing set, where the individual networks were respectively created by negative correlation learning and independent training. It is evident from Table 6 that different individual networks created by negative correlation learning were able to specialise to different parts of the testing set. For instance, in Table 6 the sizes of both correct response sets S_2 and S_4 at $\lambda = 1.0$ were 143, but the size of their intersection $S_2 \cap S_4$ was 133. The size of $S_1 \cap S_2 \cap S_3 \cap S_4$ was only 113. In contrast, the individual networks in the ensemble created by independent training were quite similar. The sizes of correct response sets S_1, S_2, S_3 and S_4 at $\lambda = 0.0$ were from 147 to 149, while the size of their intersection set $S_1 \cap S_2 \cap S_3 \cap S_4$ reached 146. There were only three different patterns correctly classified by the four individual networks in the ensemble.

In simple averaging, all the individual networks have the same combination weights and are treated equally. However, not all the networks are equally important. Because different individual networks created by negative correlation learning were able to specialise to different parts of the testing set, only the outputs of these specialists should be considered to make the final decision about the ensemble for this part of the testing set. In this experiment, a winner-takes-all method was applied to select such networks. For each pattern of the testing set,

the output of the ensemble was only decided by the network whose output had the highest activation. Table 5 shows the average results of negative correlation learning over 25 runs using the winner-takes-all combination method. The winner-takes-all combination method improved negative correlation learning significantly because there were good and poor networks for each pattern in the testing set, and winner-takes-all selected the best one. However, it did not improve the independent training much because the individual networks created by the independent training were all similar to each other.

CONCLUSIONS

This chapter describes negative correlation learning for designing neural network ensembles. It can be regarded as one way of decomposing a large problem into smaller and specialised ones, so that each sub-problem can be dealt with by an individual neural network relatively easily. A correlation penalty term in the error function was proposed to minimise mutual information and encourage the formation of specialists in the ensemble.

Negative correlation learning has been analysed in terms of mutual information on a regression task in the different noise conditions. Unlike independent training which creates larger mutual information among the ensemble, negative correlation learning can produce smaller mutual information among the ensemble. Through minimisation of mutual information, very competitive results have been produced by negative correlation learning in comparison with independent training.

This chapter compares the decision boundaries and the correct response sets constructed by negative correlation learning and the independent training for two pattern classification problems. The experimental results show that negative correlation learning has a very good classification performance. In fact, the decision boundary formed by negative correlation learning is nearly close to the optimum decision boundary generated by the Bayes classifier.

There are, however, some issues that need resolving. No special considerations were made in optimisation of the size of the ensemble and strength parameter λ in this chapter. Evolutionary ensembles with negative correlation learning for optimisation of the size of the ensemble had been studied on the classification problems (Liu, Yao & Higuchi, 2000).

REFERENCES

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Trans. Appl. Comp.*, AC-19, 716-723.

Clemen, R. T., & Winkler, R. L. (1985). Limits for the precision and value of information from dependent sources. *Operations Research*, 33:427-442.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y. & Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, 6:1289-1301.

Drucker, H., Schapire, R. & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In Hanson, S. J., Cowan, J. D. & Giles,C. L. (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 42-49. San Mateo, CA: Morgan Kaufmann.

Friedman, J. H. (1994). An overview of predictive learning and function approximation. In V. Cherkassky, J. H. Friedman, and H. Wechsler, (Eds.), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pp. 1-61. Springer-Verlag, Heidelberg, Germany.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(10):993-1001.

Jacobs, R. A. (1997). Bias/variance analyses of mixture-of-experts architectures. *Neural Computation*, 9:369-383.

Jacobs, R. A. & Jordan, M. I. (1991). A competitive modular connectionist architecture. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, (Eds.), *Advances in Neural Information Processing Systems 3*, pp. 767-773. Morgan Kaufmann, San Mateo, CA.

Jacobs, R. A., Jordan, M. I. & Barto, A. G. (1991). Task decomposition through competition in a modular connectionist architecture: the what and where vision task. *Cognitive Science*, 15:219-250.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J. & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79-87.

Jordan, M. I. & Jacobs, R. A. (1994). Hierarchical mixtures-of-experts and the em algorithm. *Neural Computation*, 6:181-214.

Liu, Y. & Yao, X. (1998a). Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems*, 4:176-185.

Liu, Y. & Yao, X. (1998b). A cooperative ensemble learning system. *In Proceedings of the 1998 IEEE International Joint Conference on Neural Networks (IJCNN'98)*, pages 2202-2207. IEEE Press, Piscataway, NJ, USA.

Liu, Y. & Yao, X. (1999). Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):716-725.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Trans. on Evolutionary Computation*, 4(4):380-725.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. New York: McGraw Hill.

Opitz, D. W. & Shavlik, J. W. (1996). Actively searching for an effective neural network ensemble. *Connection Science*, 8:337-353.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465-471.

Rosen, B. E. (1996). Ensemble learning using decorrelated neural networks. *Connection Science*, 8:373-383.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. I, pp. 318-362. MIT Press, Cambridge, MA.

Sarkar, D. (1996). Randomness in generalization ability: A source to improve it. *IEEE Trans. on Neural Networks*, 7(3):676-685.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:197-227.

Selfridge, O. G. (1958). Pandemonium: a paradigm for learning. *Mechanisation of Thought Processes: Proceedings of a Symp*. Held at the National Physical Lab., pp. 513-526. HMSO, London.

Sharkey, A. J. C. (1996). On combining artificial neural nets. *Connection Science*, 8:299-313.

van der Lubbe, J. C. A. (1997). *Information Theory*. Cambridge: Cambridge University Press.

van der Lubbe, J. C. A. (1999). *Information Theory*. (2nd ed) Prentice-Hall International, Inc.

Wallace, C. S., & Patrick, J. D. (1991). *Coding Decision Trees*. Technical Report 91/153, Department of Computer Science, Monash University, Clayton, Victoria 3168, Australia, August.

Wolpert, D. H. (1990). A mathematical theory of generalization. *Complex Systems*, 4:151–249.

Chapter II

A Perturbation Size-Independent Analysis of Robustness in Neural Networks by Randomized Algorithms

C. Alippi
Politecnico di Milano, Italy

ABSTRACT

This chapter presents a general methodology for evaluating the loss in performance of a generic neural network once its weights are affected by perturbations. Since weights represent the "knowledge space" of the neural model, the robustness analysis can be used to study the weights/performance relationship. The perturbation analysis, which is closely related to sensitivity issues, relaxes all assumptions made in the related literature, such as the small perturbation hypothesis, specific requirements on the distribution of perturbations and neural variables, the number of hidden units and a given neural structure. The methodology, based on Randomized Algorithms, allows reformulating the computationally intractable problem of robustness/sensitivity analysis in a probabilistic framework characterised by a polynomial time solution in the accuracy and confidence degrees.

INTRODUCTION

The evaluation of the effects induced by perturbations affecting a neural computation is relevant from the theoretical point of view and in developing an embedded device dedicated to a specific application.

In the first case, the interest is in obtaining a reliable and easy to be generated measure of the performance loss induced by perturbations affecting the weights of a neural network. The relevance of the analysis is obvious since weights characterise the "knowledge space" of the neural model and, hence, its inner nature. In this direction, a study of the evolution of the network's weights over training time allows for understanding the mechanism behind the generation of the knowledge space. Conversely, the analysis of a specific knowledge space (fixed configuration for weights) provides hints about the relationship between the weights space and the performance function. The latter aspect is of primary interest in recurrent neural networks where even small modifications of the weight values are critical to performance (e.g., think of the stability of an intelligent controller comprising a neural network and issues leading to robust control).

The second case is somehow strictly related to the first one and covers the situation where the neural network must be implemented in a physical device. The optimally trained neural network becomes the "golden unit" to be implemented within a finite precision representation environment as it happens in mission-critical applications and embedded systems. In these applications, behavioural perturbations affecting the weights of a neural network abstract uncertainties associated with the implementation process, such as finite precision representations (e.g., truncation or rounding in a digital hardware, fixed or low resolution floating point representations), fluctuations of the parameters representing the weights in analog solutions (e.g., associated with the production process of a physical component), ageing effects, or more complex and subtle uncertainties in mixed implementations.

The sensitivity/robustness issue has been widely addressed in the neural network community with a particular focus on specific neural topologies.

More in detail, when the neural network is composed of linear units, the analysis is straightforward and the relationship between perturbations and the induced performance loss can be obtained in a closed form (Alippi & Briozzo, 1998). Conversely, when the neural topology is non-linear, which is mostly the case, several authors assume the small perturbation hypothesis or particular hypothesis about the stochastic nature of the neural computation. In both cases, the assumptions make the mathematics more amenable with the positive consequence that a relationship between perturbations and performance loss can be derived (e.g., see Alippi & Briozzo, 1998; Pichè, 1995). Unfortunately, these analyses introduce hypotheses which are not always satisfied in all real applications.

Another classic approach requires expanding with Taylor the neural computation around the nominal value of the trained weights. A subsequent linear arised analysis follows which allows for solving the sensitivity issue (e.g., Pichè, 1995). Anyway, the validity of such approaches depend, in turn, on the validity of the small perturbation hypothesis: how to understand a priori if a perturbation is small for a given application?

In other applications the small perturbation hypothesis cannot be accepted being the involved perturbations everything but small. As an example we have the development of a digital embedded system. There, the designer has to reduce as possible the dimension of the weights by saving bits; this produces a positive impact on cost, memory size and power consumption of the final device.

Differently, other authors avoid the small perturbation assumption by focusing the attention on very specific neural network topologies and/or introducing particular assumptions regarding the distribution of perturbations, internal neural variables and inputs (Stevenson, Winter & Widrow, 1990; Alippi, Piuri & Sami, 1995).

Other authors have considered the sensitivity analysis under the small perturbation hypothesis to deal with implementation aspects. In this case, perturbations are specifically related to finite precision representations of the interim variables characterising the neural computation (Holt & Hwang, 1993; Dundar & Rose, 1995).

Differently from the limiting approaches provided in the literature, this chapter suggests a robustness/sensitivity analysis in the large, i.e., without assuming constraints on the size or nature of the perturbation; as such, small perturbation situations become only a subcase of the theory. The analysis is general and can be applied to all neural topologies, both static and recurrent in order to quantify the performance loss of the neural model when perturbations affect the model's weights.

The suggested sensitivity/robustness analysis can be applied to *All* neural network models involved in system identification, control signal/image processing and automation-based applications without any restriction. In particular, the analysis allows for solving the following problems:

- Quantify the robustness of a generically trained neural network by means of a suitable, easily to be computed and reliable robustness index;
- Compare different neural networks, solving a given application by ranking them according to their robustness;
- Investigate the criticality of a recurrent model ("stability" issue) by means of its robustness index;

- Study the efficacy and effectiveness of techniques developed to improve the robustness degree of a neural network by inspecting the improvement in robustness.

The key elements of the perturbation analysis are Randomised Algorithms – RAs- (Vidyasagar, 1996, 1998; Tempo & Dabbene, 1999; Alippi, 2002), which transform the computationally intractable problem of evaluating the robustness of a generic neural network with respect to generic, continuous perturbations, in a tractable problem solvable with a polynomial time algorithm by resorting to probability.

The increasing interest and the extensive use of Randomised Algorithms in control theory, and in particular in the robust control area (Djavan, Tulleken, Voetter, Verbruggen & Olsder, 1989; Battarcharyya, Chapellat & Keel, 1995; Bai & Tempo, 1997; Chen & Zhou, 1997; Vidyasagar, 1998; Tempo & Dabbene, 1999, Calafiore, Dabbene & Tempo, 1999), make this versatile technique extremely interesting also for the neural network researcher.

We suggest the interested reader to refer to Vidyasagar (1998) and Tempo and Dabbene (1999) for a deep analysis of the use of RAs in control applications; the author forecasts an increasing use of Randomised Algorithms in the analysis and synthesis of intelligent controllers in the neural network community.

The structure of the chapter is as follows. We first formalise the concept of robustness by identifying a natural and general index for robustness. Randomised Algorithms are then briefly introduced to provide a comprehensive analysis and adapted to estimate the robustness index. Experiments then follow to shed light on the use of the theory in identifying the robustness index for static and recurrent neural models.

A GENERAL ROBUSTNESS/SENSITIVITY ANALYSIS FOR NEURAL NETWORKS

In the following we consider a *generic* neural network implementing the $\hat{y} = f(\hat{\theta}, x)$ function where $\hat{\theta}$ is the weight (and biases) vector containing all the trained free parameters of the neural model.

In several neural models, and in particular in those related to system identification and control, the relationship between the inputs and the output of the system are captured by considering a regressor vector φ, which contains a limited time-window of actual and past inputs, outputs and possibly predicted outputs.

Of particular interest, in the zoo of neural models, are those which can be represented by means of the model structures $\hat{y}(t) = f(\varphi)$ where function $f(\cdot)$ is a regression-type neural network, characterised by N_φ inputs, N_h non-linear hidden units and a single effective linear/non-linear output (Ljung, 1987; Hertz, Krog & Palmer, 1991; Hassoun, 1995; Ljung, Sjoberg & Hjalmarsson, 1996).

The absence/presence of a dynamic in the system can be modelled by a suitable number of delay elements (or time lags), which may affect inputs (time history on external inputs u) system outputs (time history on $y(t)$) on predicted outputs (time history on $\hat{y}(t)$) or residuals (time history on $e(t) = \hat{y}(t) - y(t)$). Where it is needed $y(t)$, $\hat{y}(t)$ and $e(t)$ are vectorial entities, a component for each independent distinct variable.

Several neural model structures have been suggested in the literature, which basically differ in the regressor vector. Examples are, NARMAX and NOE topologies. NARMAX structure can be obtained by considering both past inputs and outputs of the system to infer $y(t)$. We have:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u), y(t-1), \cdots, y(t-n_y), \ldots, e(t-1), \cdots, e(t-n_e)\right]$$

Differently, the NOE structure processes only past inputs and predicted outputs, i.e.:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u), \hat{y}(t-1), \cdots, \hat{y}(t-n_y)\right].$$

Static neural networks, such as classifiers, can be obtained by simply considering external inputs:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u)\right].$$

Of course, different neural models can be considered, e.g., fully recurrent and well fit with the suggested robustness analysis.

A general, perturbation size independent, model-independent robustness analysis requires the evaluation of the loss in performance induced by a generic perturbation, in our analysis affecting the weights of a generic neural network. We denote by $y_\Delta(x) = f_\Delta(\theta, \Delta, x)$ the mathematical description of the perturbed computation and by $\Delta \in D \subseteq \Re^p$ a generic p-dimensional perturbation vector, a component for each independent perturbation affecting the neural computation $\hat{y}(t)$. The perturbation space D is characterised in stochastic terms by providing the probability density function pdf_D.

To measure the discrepancy between $y_\Delta(x)$ and $y(t)$ or $\hat{y}(t)$, we consider a generic loss function $U(\Delta)$. In the following we only assume that such performance loss function is measurable according to Lebesgue with respect to D. Lebesgue measurability for $U(\Delta)$ allows us for taking into account an extremely large class of loss functions.

Common examples for U are the Mean Square Error—MSE—loss functions

$$U(\Delta)=\frac{1}{N_x}\sum_{i=1}^{N_x}(\hat{y}(x_i)-\hat{y}(x_i,\Delta))^2 \text{ and } U(\Delta)=\frac{1}{N_x}\sum_{i=1}^{N_x}(y(x_i)-\hat{y}(x_i,\Delta))^2 . \quad (1)$$

More specifically, (1)-left compares the perturbed network with $\hat{y}$, which is supposed to be the "golden" error-free unit while (1)-right estimates the performance of the error-affected (perturbed) neural network (generalisation ability of the perturbed neural model).

The formalisation of the impact of perturbation on the performance function can be simply derived:

Definition: Robustness Index

We say that a neural network is robust at level $\bar{\gamma}$ in D, when the robustness index $\bar{\gamma}$ is the minimum positive value for which

$$U(\Delta) \le \bar{\gamma}, \forall \Delta \in D, \ \forall \gamma \ge \bar{\gamma} \quad (2)$$

Immediately, from the definition of robustness index we have that a generic neural network NN_1 is more robust than NN_2 if $\bar{\gamma}_1 < \bar{\gamma}_2$ and the property holds independently from the topology of the two neural networks.

The main problem related to the determination of the robustness index $\bar{\gamma}$ is that we have to compute $U(\Delta), \forall \Delta \in D$ if we wish a tight bound. The $\bar{\gamma}$ -identification problem is therefore intractable from a computational point of view if we relax all assumptions made in the literature as we do.

To deal with the computational aspect we associate a dual probabilistic problem to (2):

Robustness Index: Dual Problem We say that a neural network is robust at level $\bar{\gamma}$ in D with confidence η, when $\bar{\gamma}$ is the minimum positive value for which

$$\Pr(U(\Delta) \le \bar{\gamma}) \ge \eta \quad \text{holds} \quad \forall \Delta \in D, \ \forall \gamma \ge \bar{\gamma} \quad (3)$$

The probabilistic problem is weaker than the deterministic one since it tolerates the existence of a set of perturbations (whose measure according to Lebesgue is 1-η) for which $u(\Delta) > \bar{\gamma}$. In other words, not more than 100η % of perturbations $\Delta \in D$ will generate a loss in performance larger than $\bar{\gamma}$.

Probabilistic and deterministic problems are "close" to each other when we choose, as we do, $\eta = 1$. Note that $\bar{\gamma}$ depends only on the size of D and the neural network structure.

The non-linearity with respect to Δ and the lack of a priori assumptions regarding the neural network do not allow computing (2) in a closed form for the general perturbation case. The analysis, which would imply testing $U\Delta$ in correspondence with a continuous perturbation space, can be solved by resorting to probability according to the dual problem and by applying Randomised Algorithms to solve the robustness/sensitivity problem.

RANDOMIZED ALGORITHMS AND PERTURBATION ANALYSIS

In this paragraph we briefly review the theory behind Randomised Algorithms and adapt them to the robustness analysis problem.

In the following we denote by $p_\gamma = \Pr\{U(\Delta) \le \gamma\}$ the probability that the loss in performance associated with perturbations in D is below a given—but arbitrary—value γ.

Probability p_γ is unknown, cannot be computed in a close form for a generic U function and neural network topology, and its evaluation requires exploration of the whole perturbation space D.

The unknown probability p_γ can be estimated by sampling D with N independent and identically distributed samples Δ_i; extraction must be carried out according to the *pdf* of the perturbation.

For each sample Δ_i we then generate the triplet

$$\{\Delta_i, U(\Delta_i), I(\Delta_i)\}, i = 1, N \text{ where } I(\Delta_i) = \begin{cases} 1 \text{ if } U(\Delta_i) \le \gamma \\ 0 \text{ if } U(\Delta_i) > \gamma \end{cases} \quad (4)$$

The true probability p_γ can now simply be estimated as

$$\hat{p}_N = \frac{1}{N} \sum_{i=1}^{N} I(\Delta_i) \quad (5)$$

Of course, when N tends to infinity, $\hat{p}_N$ converges to p_γ. Conversely, on a finite data set of cardinality N, the discrepancy between $\hat{p}_N$ and p_γ exists and can be

simply measured as $|p_\gamma - \hat{p}_N|$. $|p_\gamma - \hat{p}_N|$ is a random variable which depends on the particular extraction of the N samples since different extractions of N samples from D will provide different estimates for $\hat{p}_N$. By introducing an accuracy degree ε on $|p_\gamma - \hat{p}_N|$ and a confidence level $1-\delta$ (which requests that the $|p_\gamma - \hat{p}_N| \le \varepsilon$ inequality is satisfied at least with probability $1-\delta$), our problem can be formalised by requiring that the inequality

$$\Pr\left\{|p_\gamma - \hat{p}_N| \le \varepsilon\right\} \ge 1 - \delta \tag{6}$$

is satisfied for $\forall \gamma \ge 0$. Of course, we wish to control the accuracy and the confidence degrees of (6) by allowing the user to choose the most appropriate values for the particular need. Finally, by extracting a number of samples from D according to the Chernoff inequality (Chernoff, 1952)

$$N \ge \frac{\ln \frac{2}{\delta}}{2\varepsilon^2} \tag{7}$$

we have that $\Pr\left\{|p_\gamma - \hat{p}_N| \le \varepsilon\right\} \ge 1 - \delta$ holds for $\forall \gamma \ge 0, \forall \delta, \varepsilon \in [0,1]$.

As an example, by considering 5% in accuracy and 99% in confidence, we have to extract 1060 samples from D; with such choice we can approximate p_γ with $\hat{p}_N$ introducing the maximum error 0.05 ($\hat{p}_N - 0.05 \le p_\gamma \le \hat{p}_N + 0.05$) and the inequality holds at least with the probability 0.99.

Other bounds can be considered instead of the Chernoff's one as suggested by Bernoulli and Bienaymè, (e.g., see Tempo & Dabbene, 1999). Nevertheless, the Chernoff's bound improves upon the others and, therefore, should be preferred if we wish to keep minimal the number of samples to be extracted. The Chernoff bound grants that:

- N is independent from the dimension of D (and hence it does not depend on the number of perturbations we are considering in the neural network);
- N is linear in $\ln \frac{1}{\delta}$ and $\frac{1}{\varepsilon^2}$ (hence it is polynomial in the accuracy and confidence degrees).

As a consequence, the dual probabilistic problem related to the identification of the robustness index $\bar{\gamma}$ can be solved with randomised algorithms and therefore with a polynomial complexity in the accuracy and the confidence degrees independently from the number of weights of the neural model network. In fact, by expanding the (6) we have that

$$\Pr\left\{p_\gamma - \hat{p}_N \le \varepsilon\right\} \ge 1-\delta \equiv \Pr\left\{\left|\Pr(u(\Delta)\le\gamma) - \frac{1}{N}\sum_i I(\Delta_i)\right| \le \varepsilon\right\} \ge 1-\delta \quad (8)$$

If accuracy ε and confidence δ are small enough, we can confuse p_γ and $\hat{p}_N$ by committing a small error. As a consequence, the dual probabilistic problem requiring $p_\gamma \ge \eta$ becomes $\hat{p}_N \ge \eta$. We surely assume ε and δ to be small enough in subsequent derivations.

The final algorithm, which allows for testing the robustness degree $\bar{\gamma}$ of a neural network, is:

1. *Select ε and δ sufficiently small to have enough accuracy and confidence.*
2. *Extract from D, according to its pdf, a number of perturbations N as suggested by (7).*
3. *Generate the indicator function $I(\Delta)$ and generate the estimate $\hat{p}_N = \hat{p}_N(\gamma)$ according to (5).*
4. *Select the minimum value γ_η from the $\hat{p}_N = \hat{p}_N(\gamma)$ function so that $\hat{p}_N(\gamma_\eta) = 1$ is satisfied $\forall \gamma \ge \gamma_\eta$. γ_η is the estimate of the robustness index $\bar{\gamma}$.*

Note that with a simple algorithm we are able to estimate in polynomial time the robustness degree $\bar{\gamma}$ of a generic neural network. The accuracy in estimating $\bar{\gamma}$ can be made arbitrarily good at the expense of a larger number of samples as suggested by Chernoff's bound.

APPLYING THE METHODOLOGY TO STUDY THE ROBUSTNESS OF NEURAL NETWORKS

In the experimental section we show how the robustness index for neural networks can be computed and how it can be used to characterise a neural model. After having presented and experimentally justified the theory supporting Randomised Algorithms, we will focus on the following problems:

- test the robustness of a given static neural network (robustness analysis);
- study the relationships between the robustness of a static neural network and the number of hidden units (structure redundancy);
- analyse the robustness of recurrent neural networks (robustness/stability analysis).

In the following experiments we consider perturbations affecting weights and biases of a neural network defined in D and subject to uniform distributions. Here,

a perturbation Δ_i affecting a generic weight w_i must be intended as a relative perturbation with respect to the weight magnitude according to the multiplicative perturbation model $w_{i,p} = w_i(1+\Delta_i), \forall i = 1, n$. A $t\%$ perturbation implies that Δ_i is drawn from a symmetrical uniform distribution of extremes

$$\left[-\frac{t}{100}, \frac{t}{100}\right];$$

a 5% perturbation affecting weights and biases composing vector $\hat{\theta}$ implies that each weight/bias is affected by an independent perturbation extracted from the [-0.05, 0.05] interval and applied to the nominal value according to the multiplicative perturbation model.

Experiment 1: *The impact of ε, δ and N on the evaluation of the robustness index*

The reference application to be learned is the simple error-free function

$$y = -x \cdot sin(x^2) + \frac{e^{-0.23 \cdot x}}{1+x^4}, \quad x \in [-3,3]$$

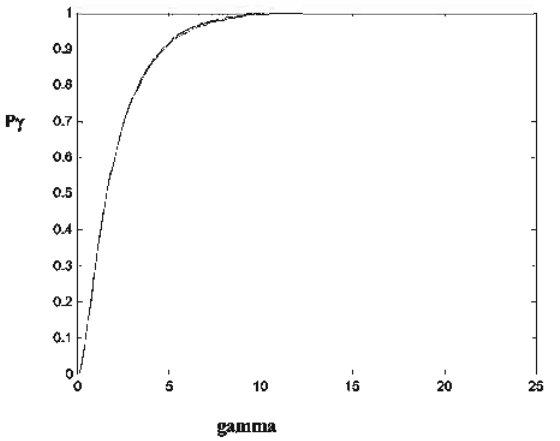
A set of 41 training data have been extracted from the function domain according to a uniform distribution. We considered static feedforward neural networks with hidden units characterised by a hyperbolic tangent activation function and a single linear output. Training was accomplished by considering a Levenberg-Marquardt algorithm applied to an MSE training function; a test set was considered during the training phase to determine the optimal stopping point so as to monitor the upsurgence of overfitting effects.

We discovered that all neural networks with at least 6 hidden units are able to solve the function approximation task with excellent performance.

In this experiment we focus the attention on the neural network characterised by 10 hidden units. After training we run the robustness algorithm by considering 7.5% of perturbations (weights are affected by perturbations up to 7.5% of their magnitude) and we chose $\varepsilon = 0.02$ and $\delta = 0.01$ from which we have to extract N=6624 samples from D. We carried out three extractions of N samples and, for each set, we computed the related $\hat{p}_N = \hat{p}_N(\gamma)$ curve.

The $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Figure 1. As we can see the curves are very close to each other. In fact, we know from the theory, that the estimated probability belongs to a neighbourhood of the true one according to the

Figure 1: $\hat{p}_N = \hat{p}_N(\gamma)$ for three different runs



$\hat{p}_N - 0.02 \le p_\gamma \le \hat{p}_N + 0.02$ relationship. A single curve is therefore enough to characterise the robustness of the envisaged neural network and there is no need to consider multiple runs. By inspecting Figure 1 we obtain that the estimate of the robustness index $\bar{\gamma}$ is γ_η=11 which implies that $U(\Delta) \le 11, \forall \Delta \in D$ with high probability.

We wish now to study the impact of N on $\hat{p}_N = \hat{p}_N(\gamma)$ by considering three runs with different ε and δ according to Table 1.

The $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Figure 2. It is interesting to note, at least for the specific application, that even with low values of N, the estimates for $\hat{p}_N = \hat{p}_N(\gamma)$ and γ_η are reasonable and not far from each other. We should anyway extract the number of samples according to Chernoff's inequality.

Experiment 2: *Testing the robustness of a given neural network*

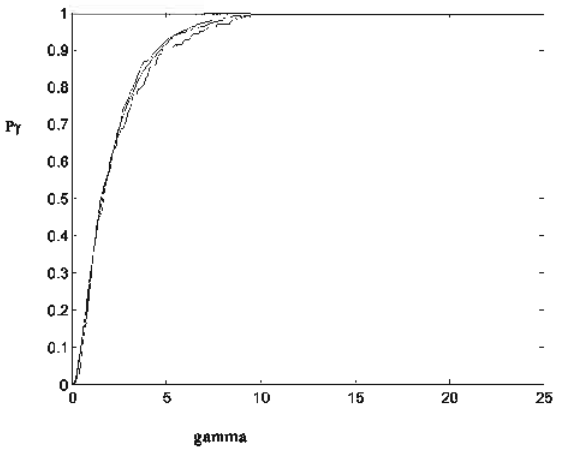
In the second experiment we test the robustness of the 10 hidden units network by considering its behaviour once affected by stronger perturbations (larger D) and, in particular, for perturbations 1%, 3%, 5%, 10%, 30%. We selected ε=0.02 and δ= 0.01.

The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ function corresponding to the different perturbations is given in Figure 3.

Table 1: ε, δ and N

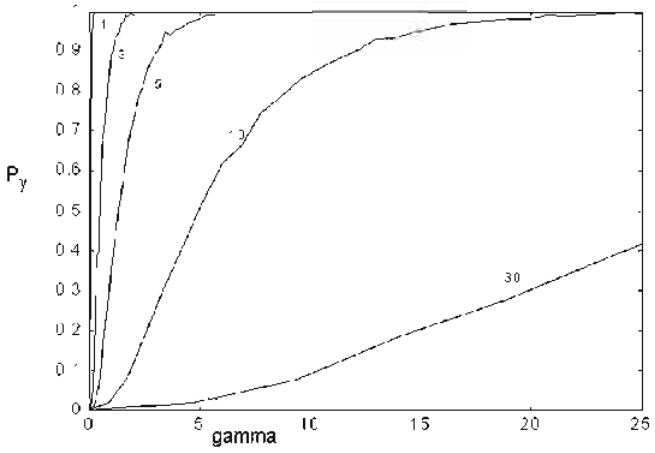
ε	δ	N
0.02	0.01	6624
0.05	0.05	738
0.1	0.1	150

Figure 2: $\hat{p}_N = \hat{p}_N(\gamma)$ for different runs with parameters given in Table 1



Again, from its definition, $\bar{\gamma}$ is the smallest value for which $\hat{p}_\gamma = 1$, $\gamma \geq \bar{\gamma}$; as an example, if we consider the 5% perturbation case, $\bar{\gamma}$ assumes a value around 7. It is obvious, but interesting to point out that, by increasing the strength of perturbation (i.e., by enlarging the extremes of the uniform distribution characterising the *pdf* of D), $\bar{\gamma}$ increases. In fact, stronger perturbations have a worse impact on the performance loss function since the error-affected neural network diverges from the error-free one. Conversely, we see that small perturbations, e.g., the 1% one, induce a very small loss in performance since the robustness index γ_η is very small.

Figure 3: $\hat{p}_\gamma$ as a function of γ for the 10 hidden units neural network



Experiment 3: *Testing the robustness of a hierarchy of performance-equivalent neural networks*

Once we have identified the robustness degree of a neural network solving an application, we can investigate whether it is possible to improve the robustness degree of the application by considering a sort of structural redundancy or not. This issue can be tackled by considering the neural hierarchy M: $M_1 \subseteq M_2 ... \subseteq M_k ...$ where M_k represents a neural network with k hidden units.

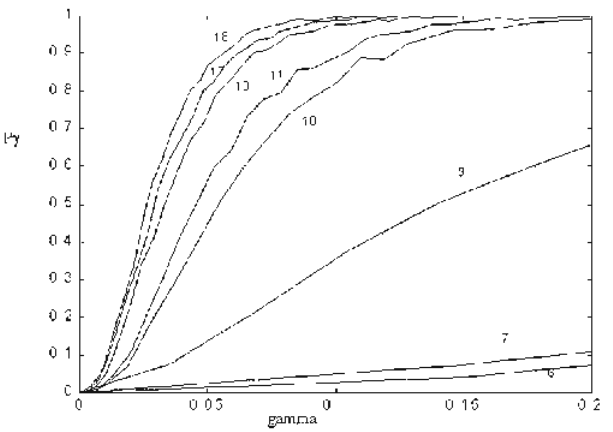
To this end, we consider a set of performance-equivalent neural networks, each of which is able to solve the application with a performance tolerable by the user. All neural networks are characterised by a different topological complexity (number of hidden units).

The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curves parameterised in the number of hidden units are given in Figure 4 in the case of 1% perturbation. We can see that by increasing the number of hidden units, $\bar{\gamma}$ decreases. We immediately realise that neural networks with a reduced number of hidden units are, for this application, less robust than the ones possessing more degrees of freedom. Large networks provide, in a way, a sort of spatial redundancy: information characterising the knowledge space of the neural networks is distributed over more degrees of freedom.

We discovered cases where a larger neural network was less robust than a smaller one: in such a case probably the complex model degenerates into a simpler one.

The evolution of $\bar{\gamma}$ over the number of hidden units parameterised with respect to the different perturbations 5%, 10% and 30% is given in Figure 5. We note that the minimal network, namely the smallest network able to solve the application, is not the more robust one for this application (in fact it possesses large values for the $\bar{\gamma}$ s).

Figure 4: $\hat{p}_\gamma$ *over* γ *and parameterised in the number of hidden units*



This trend—verified also with other applications—suggests that the robustness degree of the neural network improves on the average by increasing the number of hidden units (spatial redundancy). Anyway, with a small increase in the topological complexity (e.g., by considering the 13 hidden units model instead of the 6 one), we obtain a significant improvement according to the robustness level. There is no need to consider more complex neural networks since the improvement in robustness is small.

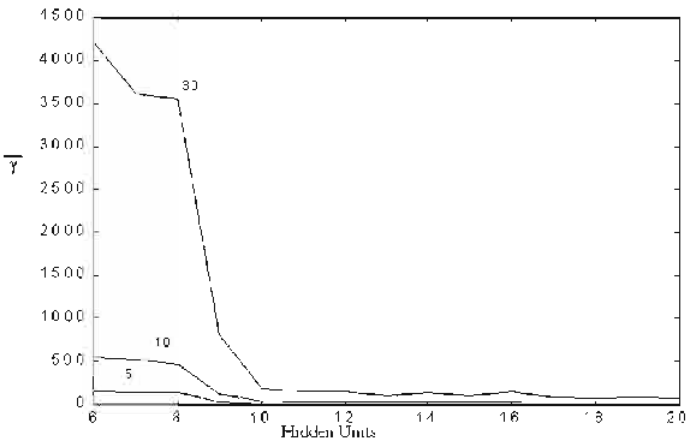
Experiment 4: *Testing the robustness of recurrent neural networks*

The goal of the last experiment is to study the robustness/stability of recurrent neural networks with the suggested theory. The chosen application refers to the identification of the open-loop stable, nonlinear, continuous system suggested in Norgaard (2000). The input and the corresponding output sequence of the system to be identified is given in Figure 6.

We first considered an NOE recurrent neural network with 5 hidden units characterised by the regressor vector $\varphi = [u(t-1), u(t-2), \hat{y}(t-1), \hat{y}(t-2)]$. The non-linear core of the neural network is a static regression type neural network as the one considered in the function approximation experiments. The topology of the NOE network is given in figure 7.

Once trained, the network we applied the methodology to estimates the robustness of the recurrent model. The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve, evaluated with $\varepsilon = \delta = 0.05$, for the 0.1% perturbation case is given in Figure 8. As we could have expected, differently from the static function approximation application, the recurrent NOE neural network is sensitive even to small perturbations affecting the knowledge space of the network.

Figure 5: $\bar{\gamma}$ *as function of the hidden units,* ε *=0.04,* δ *=0.01*



We identified the dynamic system with a NARMAX neural network characterised by 5 hidden units and the structure given in Figure 9. For such topology we selected the regressor vector

$$\varphi = [u(t-1), u(t-2), y(t-1), \cdot y(t-2), e(t-1), e(t-2)]$$

Figure 10 shows the $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve. It is interesting to note that the NARMAX neural network is less robust than the corresponding NOE model. The basic reason for such behaviour is due to the fact that the recurrent model does not receive directly as input the fed-back network output but only the residual *e*.

During training the NOE model must somehow learn more deeply the concept of stability since even small variations of weights associated with the training phase weights update would produce a trajectory diverging from the system output to be mimicked. This effect is due to the pure fed-back structure of the NOE model which

Figure 6: The input and the corresponding output of the dynamic system

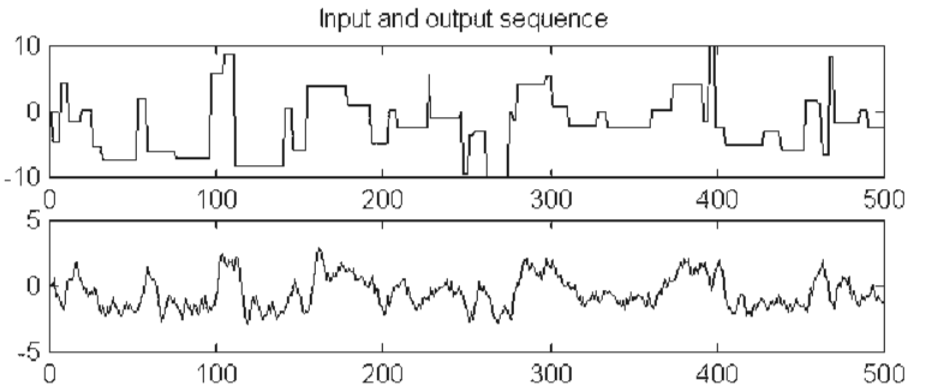


Figure 7: The considered NOE neural network

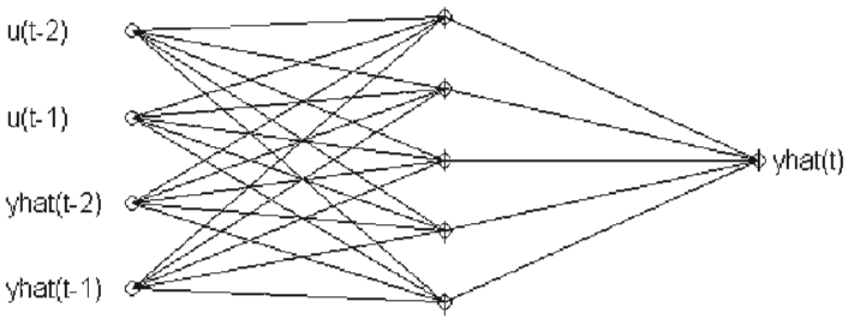


Figure 8: The $\hat{p}_\gamma$ function for the NOE neural network

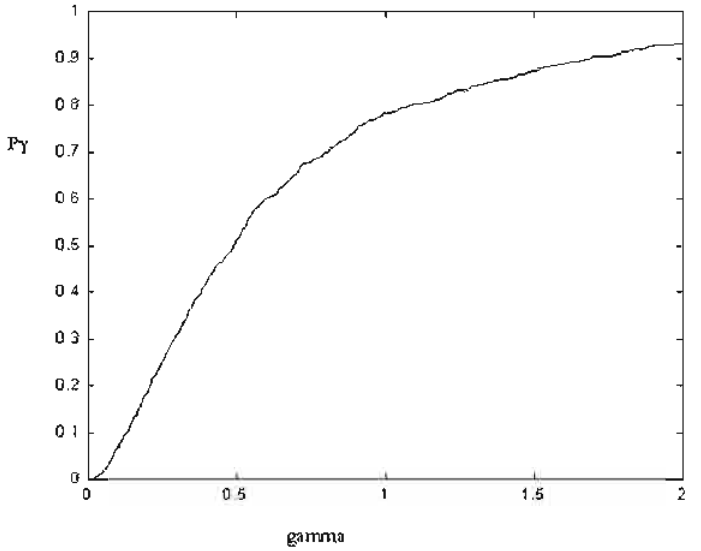
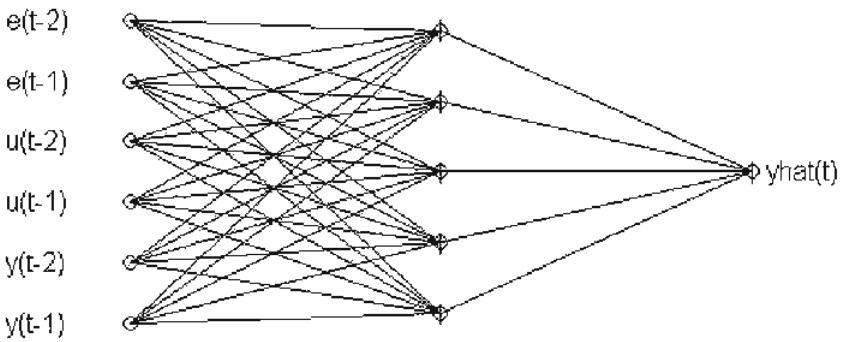
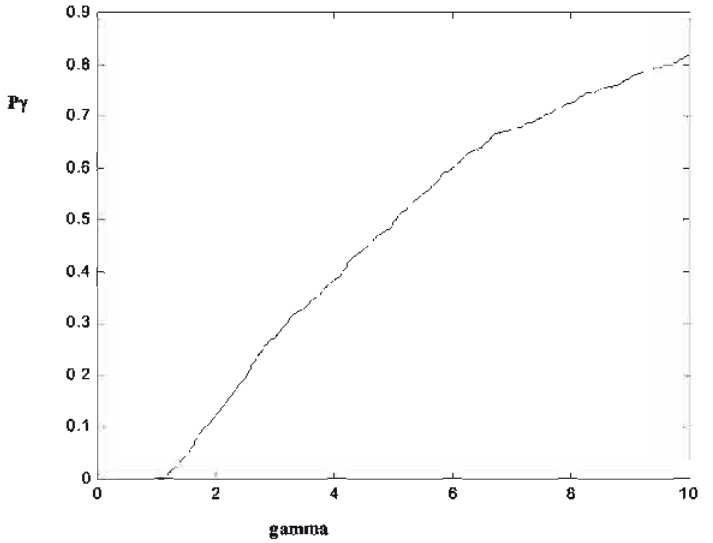


Figure 9: The considered NARMAX neural network



receives as inputs past predicted output and not direct information from the process. Interestingly, this requires the neural model to implicitly learn, during the training phase, the concept of robustness as proven by the $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve. Conversely, the NARMAX model has a smoother and less complex training phase since it receives fresh information directly from the process (y values) which help the neural model to be stable. As such, the training procedure will not search for weights configuration particularly robust since small deviations, which could make the system unstable, will be directly stabilised by the true information coming from the process.

Figure 10: The $\hat{p}_\gamma$ function for the NARMAX neural network



CONCLUSION

The main results of the chapter can be summarised as follows. Once given a trained neural network:

- the effects of perturbations affecting the network weights can be evaluated regardless of the topology and structure of the neural network, the strength of the perturbation by considering a probabilistic approach;
- the robustness/sensitivity analysis can be carried out with a Poly-time algorithm by resorting to Randomised Algorithms;
- the analysis is independent from the figure of merit considered to evaluate the loss in performance induced by the perturbations.

REFERENCES

Alippi, C. (2002). Randomized Algorithms: A system-level, Poly-time analysis of robust computation. *IEEE Transactions on Computers.* 51(7).

Alippi, C. & Briozzo, L. (1998). Accuracy vs. precision in digital VLSI architectures for signal processing. *IEEE Transactions on Computers,* 47(4).

Alippi, C., Piuri, V. & Sami, M. (1995). Sensitivity to Errors in Artificial Neural Networks: A Behavioural Approach. IEEE Transactions on Circuits and Systems: Part 1, 42(6).

Bai, E., Tempo, R. & Fu, M. (1997). Worst-case properties of the uniform distribution and randomized algorithms for robustness analysis. *IEEE-American Control Conference,* Albuquerque, NM.

Bhattacharyya, S.P, Chapellat, H. & Keel, L.H. (1995). *Robust Control: The Parametric Approach.* Englewoods Cliffs, NJ: Prentice Hall.

Calafiore, G., Dabbene, F. & Tempo, R. (1999). Uniform sample generation of vectors in lp balls for probabilistic robustness analysis. In *Recent Advances in Control,* Springer-Verlag.

Chen, X. & Zhou, K. (1997). On the probabilistic characterisation of model uncertainty and robustness. *Proceedings IEEE-36th Conference on Decision and Control,* San Diego, CA.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. 23.

Djavdan, P., Tulleken, H., Voetter, M., Verbruggen, H. & Olsder, G. (1989). Probabilistic Robust Controller Design. *Proceedings IEEE-28th Conference on Decision and Control,* Tampa, FL.

Dundar, G., Rose, K. (1995). The effects of Quantization on Multilayer Neural Networks, IEEE Transactions of Neural Networks. 6(6).

Hassoun, M.H. (1995). *Fundamentals of Artificial Neural Networks,* The MIT Press.

Hertz, J., Krogh, A. & Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation,* Addison-Wesley Publishing Co.

Holt, J. & Hwang, J. (1993). Finite precision error analysis of neural network hardware implementations. IEEE Transactions on Computers. 42(3).

Ljung, L. (1987). *System Identification, theory for the user,* Prentice-Hall.

Ljung, L. Sjoberg, J. & Hjalmarssoon, H. (1996). On neural networks model structures in system identification. In *Identification, Adaptation, Learning.* NATO ASI series.

Norgaard, M. (2000). Neural Network-Based System Identification Toolbox.

Piché, S. (1995). The selection of weights accuracies for Madalines. IEEE Transactions on Neural Networks. 6(2).

Stevenson, M., Winter, R. & Widrow, B. (1990). Sensitivity of Feedforward neural networks to weights errors, IEEE Transactions on Neural Networks. 1(1).

Tempo, R. & Dabbene, F. (1999). Probabilistic Robustness Analysis and Design of Uncertain Systems. Progress in Systems and Control Theory. 25.

Vidyasagar, M. (1996). *A Theory of Learning and Generalisation with Applications to Neural Networks and Control Systems.* Berlin: Springer-Verlag.

Vidyasagar, M. (1998). Statistical learning Theory and Randomized algorithms for Control. IEEE-Control systems.

Chapter III

Helicopter Motion Control Using a General Regression Neural Network

T.G.B. Amaral
Superior Technical School of Setúbal – IPS, Portugal

M.M. Crisóstomo
University of Coimbra, Portugal

V. Fernão Pires
Superior Technical School of Setúbal – IPS, Portugal

ABSTRACT

This chapter describes the application of a general regression neural network (GRNN) to control the flight of a helicopter. This GRNN is an adaptive network that provides estimates of continuous variables and is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. An important reason for using the GRNN as a controller is the fast learning capability and its non-iterative process. The disadvantage of this neural network is the amount of computation required to produce an estimate, which can become large if many training instances are gathered. To overcome this problem, it is described as a clustering algorithm to produce representative exemplars from a group of training instances that are close to one another reducing the computation amount to obtain an estimate. The reduction of training data used by the GRNN can make it possible to separate the obtained representative

exemplars, for example, in two data sets for the coarse and fine control. Experiments are performed to determine the degradation of the performance of the clustering algorithm with less training data. In the control flight system, data training is also reduced to obtain faster controllers, maintaining the desired performance.

INTRODUCTION

The application of a general regression neural network to control a non-linear system such as the flight of a helicopter at or near hover is described. This general regression neural network in an adaptive network that provides estimates of continuous variables and is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. The automatic flight control system, through the longitudinal and lateral cyclic, the collective and pedals are used to enable a helicopter to maintain its position fixed in space for a long period of time. In order to reduce the computation amount of the gathered data for training, and to obtain an estimate, a clustering algorithm was implemented. Simulation results are presented and the performance of the controller is analysed.

HELICOPTER MOTION CONTROL

Recently, unmanned helicopters, particularly large-scale ones, have been expected not only for the industrial fields such as agricultural spraying and aerial photography, but also for such fields as observation, rescuing and fire fighting. For monotonous and dangerous tasks, an autonomous flight control of the helicopter is advantageous.

In general, the unmanned helicopter is an example of an intelligent autonomous agent. Autonomous flight control involves some difficulties due to the following:

- it is non-linear;
- flight modes are cross-coupled;
- its dynamics are unstable;
- it is a multivariate (i.e., there are many input-output variables) system;
- it is sensitive to external disturbances and environmental conditions such as wind, temperature, etc;
- it can be used in many different flight modes (e.g., hover or forward flight), each of which requires different control laws;
- it is often used in dangerous environments (e.g., at low altitudes near obstacles).

These characteristics make the conventional control difficult and create a challenge to the design of intelligent control systems.

For example, although helicopters are non-linear systems, NN controllers are capable of controlling them because they are also inherently non-linear. The instabilities that result from time delays between changes in the system input and output can be addressed with the previous learning of the network with a set of data that represents the pilots knowledge to stabilize the helicopter. Linear NN can be implemented to compensate the cross-couplings between control inputs, mainly when the helicopter makes a significant change in its flight.

Therefore, a supervised general regression neural network can be used to control the flight modes of an unmanned helicopter. The regression is the least-mean-squares estimation of the value of a variable based on data samples. The term *general regression* implies that the regression surface is not restricted by being linear. If the values of the variables to be estimated are future values, the general regression network (GRNN) is a predictor. If they are dependent variables related to input variables in a process, system or plant, the GRNN can be used to model the process, system or plant. Once the system is modelled, a control surface can be defined in terms of samples of control variables that, given a state vector of the system, improve the output of the system. If a GRNN is trained using these samples, it can estimate the entire control surface, becoming a controller. A GRNN can be used to map from one set of sample points to another. If the target space has the same dimension as the input space, and if the mapping is one-to-one, an inverse mapping can easily be formed using the same examples. When the variables to be estimated are for intermediate values between given points, then the GRNN can be used as an interpolator.

In all cases, the GRNN instantly adapts to new data points. This could be a particular advantage for training robots to emulate a teacher or for any system whose model changes frequently.

SYSTEM MODELLING

The helicopter control is one of the popular non-linear educational control problems. Due to its highly non-linear dynamics, it gives the possibility to demonstrate basic features and limits of non-linear control concepts. Sugeno (1997, 1998) developed a fuzzy-logic based control system to replace the aircraft's normal set of control inputs. Other researchers, such has Phillips et al. (1994), Wade et al (1994), and Wade and Walker (1994), have developed fuzzy logic flight controls describing systems that include mechanisms for discovering and tuning fuzzy rules in adaptive controllers. (Larkin, 1984) described a model of an autopilot controller based on fuzzy algorithms. An alternative approach to real-time control of an

autonomous flying vehicle based on behavioral, or reactive, approach is proposed by Fagg et al. (1993). A recurrent neural network used to forward modeling of helicopter flight dynamics was described by Walker and Mo (1994). The NN-based controllers can indirectly model human cognitive performance by emulating the biological processes underlying human skill acquisition.

The main difference between NN-based controllers and conventional control systems is that, in the NN case, systems are built from indirectly representations of control knowledge similar to those employed by skilled humans, while in the conventional design case, a deep analytical understanding of the system dynamics is needed. The ability of humans to pilot manned helicopters with only the qualitative knowledge indicate that NN-based controllers with similar capabilities can also be developed.

The helicopter can be modelled as a linear system around trim points, i.e., a flight with no accelerations and no moments. The state space equations are a natural form, which can represent the helicopter motion. The general mathematical model is given by:

$$\dot{x} = Ax + Bu_c$$

$$y = Cx + Du_c$$

where x, u_c *and* y are the state vector, control vector and output vector, respectively.

The helicopter used to simulate the flight in hover position was a single main rotor helicopter of 15,000 pounds. The control and state vectors are defined as:

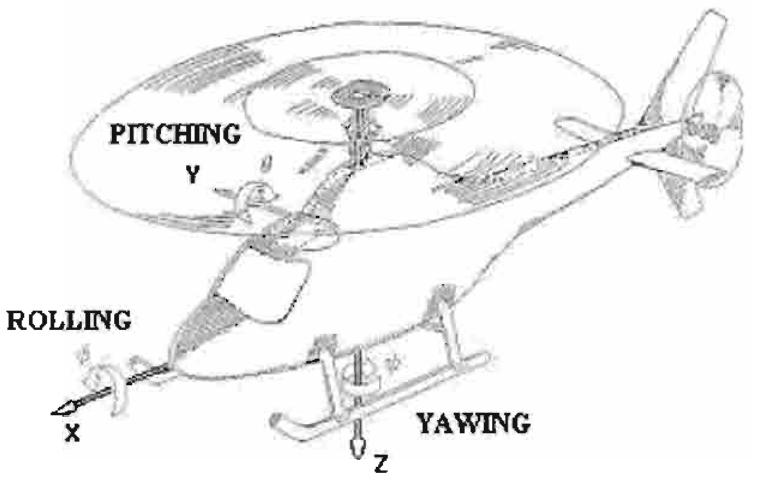
$$u_c^T = [\delta_a \ \delta_b \ \delta_c \ \delta_d] \tag{1}$$

$$x^T = [u \ v \ w \ p \ q \ r \ \phi \ \theta \ \varphi \ \mathrm{x} \ y \ z] \tag{2}$$

where

δ_a is the collective control [*inches*];
δ_b and δ_c are the longitudinal and lateral cyclic controls, respectively [*inches*];

Figure 1: Helicopter coordinates



δ_d is the pedal control [*inches*];
u, v and w are the perturbation linear velocities [*ft*/sec];
p, q and r are the perturbation angular velocities [*rad*/sec];
ϕ, θ and φ are the perturbation euler angles for roll, pitch and yaw [*rad*];
x, y and z are the perturbation linear displacements over the ground [*ft*].

Figure 1 shows the coordinate system to describe the motion of the helicopter. The origin of the helicopter axes is placed on the center of gravity.

The thrust of the main rotor, thus mainly the vertical acceleration, is controlled by the collective control (δ_a). The pitching moment, that is, nose pointing up or down, is controlled by the longitudinal cyclic control (δ_b). The rolling moment, that is, right wing tip down, left wing tip up, and vice versa, is controlled by the lateral cyclic control (δ_c). The yawing moment, that is, nose left and right, is controlled by the pedal control (δ_d).

The corresponding differential equations that represent the behavior of the helicopter in hover position are:

$$\frac{du}{dt} = -0.069u - 0.032v + 116.8p + 1168.5q - 6.15r - 32.19\theta + 0.118\delta_a$$
$$- 2.79\delta_b - 0.25\delta_c + 0.0043\delta_d$$

$$\frac{dv}{dt} = 0.017u - 0.085v - 0.0021w - 430.5p + 381.3q + 30.75r + 32.14\phi + 0.023\theta - 0.14\delta_a - \delta_b + 0.665\delta_c - 1.39\delta_d$$

$$\frac{dw}{dt} = -0.0021v - 0.257w + 7.99p + 46.74q + 135.3r + 1.85\phi - 0.404\theta - 9.23\delta_a - 0.107\delta_b - 0.01\delta_c$$

$$\frac{dp}{dt} = 0.45u - 0.687v - 0.0021w - 6027.2p + 5043.16q + 664.2r - 1.82\delta_a - 13.7\delta_b + 8.58\delta_c - 5.15\delta_d$$

$$\frac{dq}{dt} = 0.665u + 0.429v - 0.043w - 1537.5p - 15744.5q - 12.3r - 0.966\delta_a + 37.13\delta_b + 3.43\delta_c + 0.75\delta_d$$

$$\frac{dr}{dt} = -0.0214u + 0.515v + 0.0064w - 369.0p - 44.28q - 1266.9r + 25.97\delta_a - 0.15\delta_b + 0.075\delta_c + 40.78\delta_d$$

$$\frac{d\phi}{dt} = p$$

$$\frac{d\theta}{dt} = q$$

$$\frac{d\varphi}{dt} = r$$

$$\frac{dx}{dt} = u$$

$$\frac{dy}{dt} = v$$

$$\frac{dz}{dt} = w$$

Since each motion is not independent of δ_a, δ_b, δ_c and δ_d, there exists a cross-coupling.

Figure 2 shows the root locus for the model described above. Figure 2(a) shows the root locus, considering the collective control as the input and the vertical displacement as the output. In Figure 2(c), the longitudinal cyclic and the forward displacement are the input and the output, respectively. Figure 2(e) shows the root locus considering the lateral cyclic as the input and the lateral displacement as the output. Figures 2(b), (d) and (f) show the zoom of the region near the imaginary axis as well as the roots that dominate the transient response. In general, the contribution

Figure 2: Root locus of the helicopter model

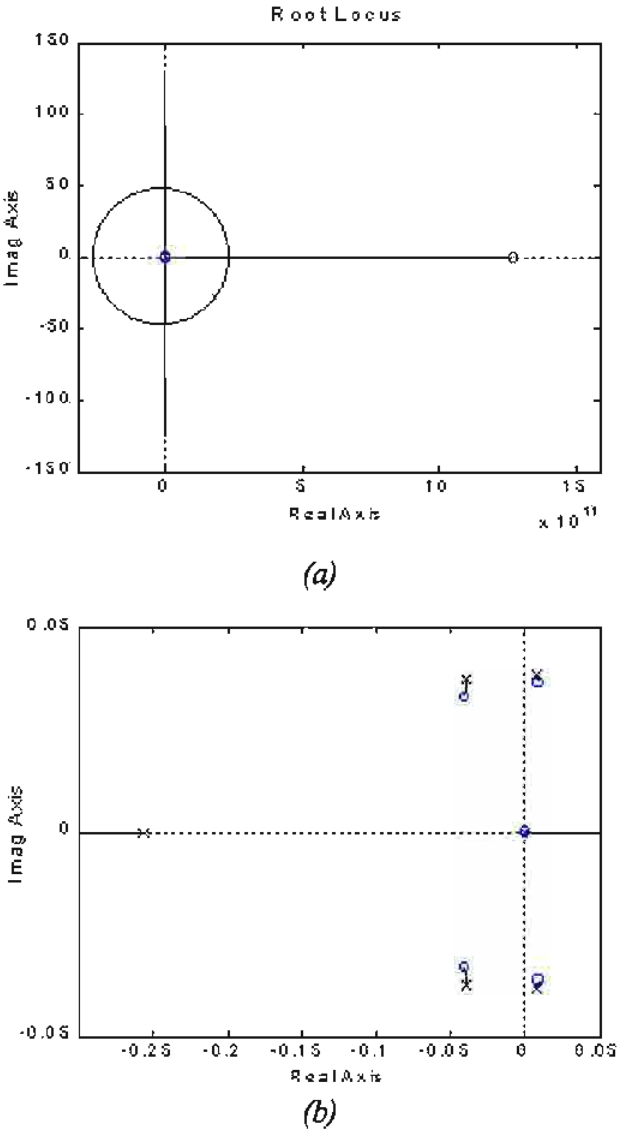
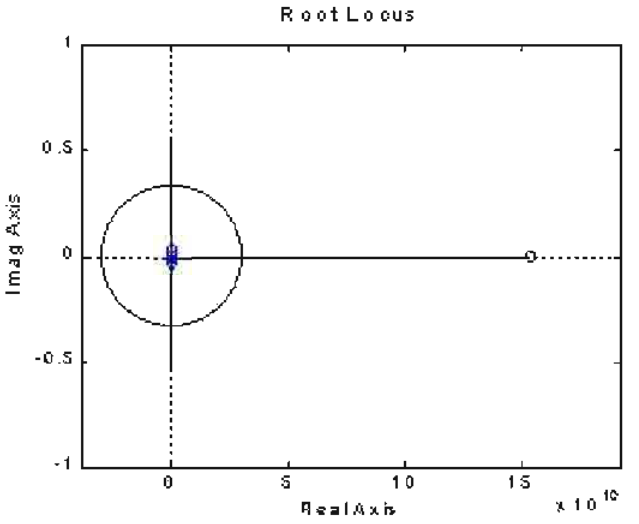
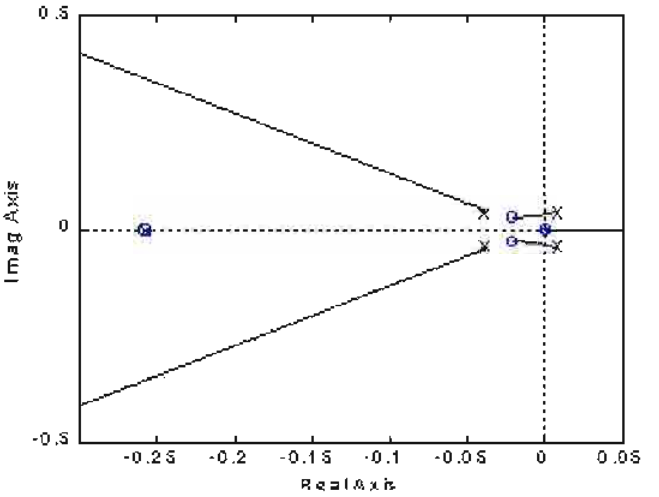


Figure 2: Root locus of the helicopter model (continued)



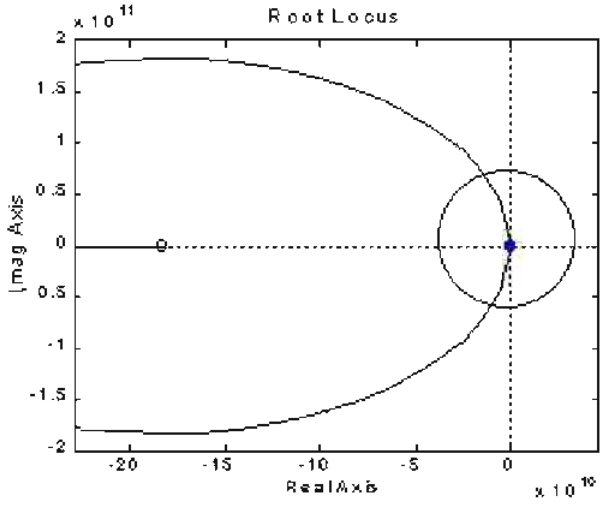
(c)



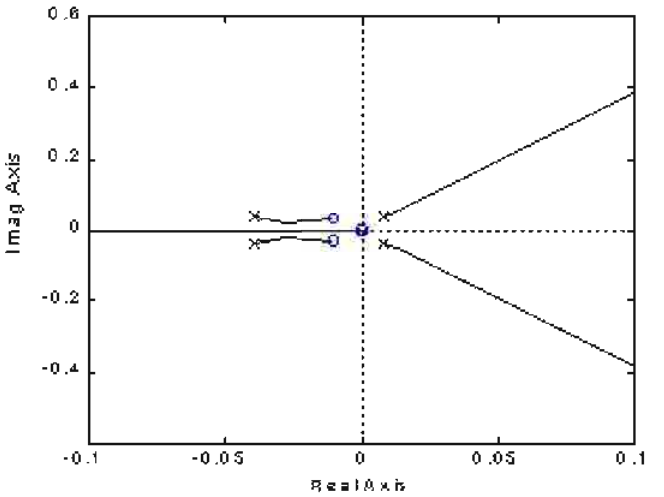
(d)

in the time response of roots that lie relatively far to the left in the s-plane will be small. These three Figures clearly show that some of the eigenvalues corresponding to the helicopter model are in the right side of the s-plane, with positive real-part values, making the system unstable.

Figure 2: Root locus of the helicopter model (continued)



(e)



(f)

GENERAL REGRESSION NEURAL NETWORK

The generalized regression neural networks are memory-based feed-forward networks originally developed in the statistics literature by Nadaraya (1964) and known as Nadaraya-Watson kernel regression. Then the GRNN was 're-discovered' by Specht (1991) and Chen, C. (1996), with the desired capability of

Computational Intelligence in Control

Masoud Mohammadian
Ruhul Amin Sarker
Xin Yao

IDEA GROUP PUBLISHING

Computational Intelligence in Control

Masoud Mohammadian, University of Canberra, Australia
Ruhul Amin Sarker, University of New South Wales, Australia
Xin Yao, University of Birmingham, UK

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore • Beijing

Acquisition Editor:	Mehdi Khosrowpour
Senior Managing Editor:	Jan Travers
Managing Editor:	Amanda Appicello
Development Editor:	Michele Rossi
Copy Editor:	Maria Boyer
Typesetter:	Tamara Gillis
Cover Design:	Integrated Book Technology
Printed at:	Integrated Book Technology

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Mohammadian, Masoud.
Computational intelligence in control / Masoud Mohammadian, Ruhul Amin Sarker and Xin Yao.
p. cm.
ISBN 1-59140-037-6 (hardcover) -- ISBN 1-59140-079-1 (ebook)
1. Neural networks (Computer science) 2. Automatic control. 3. Computational intelligence. I. Amin, Ruhul. II. Yao, Xin, 1962- III. Title.
QA76.87 .M58 2003
006.3--dc21

2002014188

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Computational Intelligence in Control

Table of Contents

Preface

This book covers the recent applications of computational intelligence techniques for modelling, control and automation. The application of these techniques has been found useful in problems when the process is either difficult to model or difficult to solve by conventional methods. There are numerous practical applications of computational intelligence techniques in modelling, control, automation, prediction, image processing and data mining.

Research and development work in the area of computational intelligence is growing rapidly due to the many successful applications of these new techniques in very diverse problems. "Computational Intelligence" covers many fields such as neural networks, (adaptive) fuzzy logic, evolutionary computing, and their hybrids and derivatives. Many industries have benefited from adopting this technology. The increased number of patents and diverse range of products developed using computational intelligence methods is evidence of this fact.

These techniques have attracted increasing attention in recent years for solving many complex problems. They are inspired by nature, biology, statistical techniques, physics and neuroscience. They have been successfully applied in solving many complex problems where traditional problem-solving methods have failed. These modern techniques are taking firm steps as robust problem-solving mechanisms.

This volume aims to be a repository for the current and cutting-edge applications of computational intelligent techniques in modelling control and automation, an area with great demand in the market nowadays.

With roots in modelling, automation, identification and control, computational intelligence techniques provide an interdisciplinary area that is concerned with learning and adaptation of solutions for complex problems. This instantiated an enormous amount of research, searching for learning methods that are capable of controlling novel and non-trivial systems in different industries.

This book consists of open-solicited and invited papers written by leading researchers in the field of computational intelligence. All full papers have been peer review by at least two recognised reviewers. Our goal is to provide a book

that covers the foundation as well as the practical side of the computational intelligence.

The book consists of 17 chapters in the fields of self-learning and adaptive control, robotics and manufacturing, machine learning, evolutionary optimisation, information retrieval, fuzzy logic, Bayesian systems, neural networks and hybrid evolutionary computing.

This book will be highly useful to postgraduate students, researchers, doctoral students, instructors, and partitioners of computational intelligence techniques, industrial engineers, computer scientists and mathematicians with interest in modelling and control.

We would like to thank the senior and assistant editors of Idea Group Publishing for their professional and technical assistance during the preparation of this book. We are grateful to the unknown reviewers for the book proposal for their review and approval of the book proposal. Our special thanks goes to Michele Rossi and Mehdi Khosrowpour for their assistance and their valuable advise in finalizing this book.

We would like to acknowledge the assistance of all involved in the collation and review process of the book, without whose support and encouragement this book could not have been successfully completed.

We wish to thank all the authors for their insights and excellent contributions to this book. We would like also to thank our families for their understanding and support throughout this book project.

M. Mohammadian, R. Sarker and X. Yao

SECTION I:

NEURAL NETWORKS DESIGN, CONTROL AND ROBOTICS APPLICATION

Chapter I

Designing Neural Network Ensembles by Minimising Mutual Information

Yong Liu
The University of Aizu, Japan

Xin Yao
The University of Birmingham, UK

Tetsuya Higuchi
National Institute of Advanced Industrial Science and Technology, Japan

ABSTRACT

This chapter describes negative correlation learning for designing neural network ensembles. Negative correlation learning has been firstly analysed in terms of minimising mutual information on a regression task. By minimising the mutual information between variables extracted by two neural networks, they are forced to convey different information about some features of their input. Based on the decision boundaries and correct response sets, negative correlation learning has been further studied on two pattern classification problems. The purpose of examining the decision boundaries and the correct response sets is not only to illustrate the learning behavior of negative correlation learning, but also to cast light on how to design more effective neural network ensembles. The experimental results showed the decision boundary of the trained neural network ensemble by negative correlation learning is almost as good as the optimum decision boundary.

INTRODUCTION

In single neural network methods, the neural network learning problem is often formulated as an optimisation problem, i.e., minimising certain criteria, e.g., minimum error, fastest learning, lowest complexity, etc., about architectures. Learning algorithms, such as backpropagation (BP) (Rumelhart, Hinton & Williams, 1986), are used as optimisation algorithms to minimise an error function. Despite the different error functions used, these learning algorithms reduce a learning problem to the same kind of optimisation problem.

Learning is different from optimisation because we want the learned system to have best generalisation, which is different from minimising an error function. The neural network with the minimum error on the training set does not necessarily have the best generalisation unless there is an equivalence between generalisation and the error function. Unfortunately, measuring generalisation exactly and accurately is almost impossible in practice (Wolpert, 1990), although there are many theories and criteria on generalisation, such as the minimum description length (Rissanen, 1978), Akaike's information criteria (Akaike, 1974) and minimum message length (Wallace & Patrick, 1991). In practice, these criteria are often used to define better error functions in the hope that minimising the functions will maximise generalisation. While better error functions often lead to better generalisation of learned systems, there is no guarantee. Regardless of the error functions used, single network methods are still used as optimisation algorithms. They just optimise different error functions. The nature of the problem is unchanged.

While there is little we can do in single neural network methods, there are opportunities in neural network ensemble methods. Neural network ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, a neural network ensemble combines a set of neural networks which learn to subdivide the task and thereby solve it more efficiently and elegantly. A neural network ensemble offers several advantages over a monolithic neural network. First, it can perform more complex tasks than any of its components (i.e., individual neural networks in the ensemble). Secondly, it can make an overall system easier to understand and modify. Finally, it is more robust than a monolithic neural network and can show graceful performance degradation in situations where only a subset of neural networks in the ensemble are performing correctly. Given the advantages of neural network ensembles and the complexity of the problems that are beginning to be investigated, it is clear that the neural network ensemble method will be an important and pervasive problem-solving technique.

The idea of designing an ensemble learning system consisting of many subsystems can be traced back to as early as 1958 (Selfridge, 1958; Nilsson, 1965). Since the early 1990s, algorithms based on similar ideas have been developed in many different but related forms, such as neural network ensembles

(Hansen & Salamon, 1990; Sharkey, 1996), mixtures of experts (Jacobs, Jordan, Nowlan & Hinton, 1991; Jacobs & Jordan, 1991; Jacobs, Jordan & Barto, 1991; Jacobs, 1997), various boosting and bagging methods (Drucker, Cortes, Jackel, LeCun & Vapnik, 1994; Schapire, 1990; Drucker, Schapire & Simard, 1993) and many others. There are a number of methods of designing neural network ensembles. To summarise, there are three ways of designing neural network ensembles in these methods: independent training, sequential training and simultaneous training.

A number of methods have been proposed to train a set of neural networks independently by varying initial random weights, the architectures, the learning algorithm used and the data (Hansen et al., 1990; Sarkar, 1996). Experimental results have shown that networks obtained from a given network architecture for different initial random weights often correctly recognize different subsets of a given test set (Hansen et al., 1990; Sarkar, 1996). As argued in Hansen et al. (1990), because each network makes generalisation errors on different subsets of the input space, the collective decision produced by the ensemble is less likely to be in error than the decision made by any of the individual networks.

Most independent training methods emphasised independence among individual neural networks in an ensemble. One of the disadvantages of such a method is the loss of interaction among the individual networks during learning. There is no consideration of whether what one individual learns has already been learned by other individuals. The errors of independently trained neural networks may still be positively correlated. It has been found that the combining results are weakened if the errors of individual networks are positively correlated (Clemen & Winkler, 1985). In order to decorrelate the individual neural networks, sequential training methods train a set of networks in a particular order (Drucker et al., 1993; Opitz & Shavlik, 1996; Rosen, 1996). Drucker et al. (1993) suggested training the neural networks using the boosting algorithm. The boosting algorithm was originally proposed by Schapire (1990). Schapire proved that it is theoretically possible to convert a weak learning algorithm that performs only slightly better than random guessing into one that achieves arbitrary accuracy. The proof presented by Schapire (1990) is constructive. The construction uses filtering to modify the distribution of examples in such a way as to force the weak learning algorithm to focus on the harder-to-learn parts of the distribution.

Most of the independent training methods and sequential training methods follow a two-stage design process: first generating individual networks, and then combining them. The possible interactions among the individual networks cannot be exploited until the integration stage. There is no feedback from the integration stage to the individual network design stage. It is possible that some of the independently designed networks do not make much contribution to the integrated system. In

order to use the feedback from the integration, simultaneous training methods train a set of networks together. Negative correlation learning (Liu & Yao, 1998a, 1998b, 1999) and the mixtures-of-experts (ME) architectures (Jacobs et al., 1991; Jordan & Jacobs, 1994) are two examples of simultaneous training methods. The idea of negative correlation learning is to encourage different individual networks in the ensemble to learn different parts or aspects of the training data, so that the ensemble can better learn the entire training data. In negative correlation learning, the individual networks are trained simultaneously rather than independently or sequentially. This provides an opportunity for the individual networks to interact with each other and to specialise.

In this chapter, negative correlation learning has been firstly analysed in terms of minimising mutual information on a regression task. The similarity measurement between two neural networks in an ensemble can be defined by the explicit mutual information of output variables extracted by two neural networks. The mutual information between two variables, output F_i of network i and output F_j of network j, is given by

$$I(F_i; F_j) = h(F_i) + h(F_j) - h(F_i, F_j) \tag{1}$$

where $h(F_i)$ is the entropy of F_i, $h(F_j)$ is the entropy of F_j, and $h(F_i, F_j)$ is the joint differential entropy of F_i and F_j. The equation shows that joint differential entropy can only have high entropy if the mutual information between two variables is low, while each variable has high individual entropy. That is, the lower mutual information two variables have, the more different they are. By minimising the mutual information between variables extracted by two neural networks, they are forced to convey different information about some features of their input. The idea of minimising mutual information is to encourage different individual networks to learn different parts or aspects of the training data so that the ensemble can learn the whole training data better.

Based on the decision boundaries and correct response sets, negative correlation learning has been further studied on two pattern classification problems. The purpose of examining the decision boundaries and the correct response sets is not only to illustrates the learning behavior of negative correlation learning, but also to cast light on how to design more effective neural network ensembles. The experimental results showed the decision boundary of the trained neural network ensemble by negative correlation learning is almost as good as the optimum decision boundary.

The rest of this chapter is organised as follows: Next, the chapter explores the connections between the mutual information and the correlation coefficient, and

explains how negative correlation learning can be used to minimise mutual information; then the chapter analyses negative correlation learning via the metrics of mutual information on a regression task; the chapter then discusses the decision boundaries constructed by negative correlation learning on a pattern classification problem; finally the chapter examines the correct response sets of individual networks trained by negative correlation learning and their intersections, and the chapter concludes with a summary of the chapter and a few remarks.

MINIMISING MUTUAL INFORMATION BY NEGATIVE CORRELATION LEARNING

Minimisation of Mutual Information

Suppose the output F_i of network i and the output F_j of network j are Gaussian random variables. Their variances are σ_i^2 and σ_j^2, respectively. The mutual information between F_i and F_j can be defined by Eq.(1) (van der Lubbe, 1997, 1999). The differential entropy $h(F_i)$ and $h(F_j)$ are given by

$$h(F_i) = [1 + log(2\pi\sigma_i^2)] / 2 \tag{2}$$

and

$$h(F_j) = [1 + log(2\pi\sigma_j^2)] / 2 \tag{3}$$

The joint differential entropy $h(F_i, F_j)$ is given by

$$h(F_i, F_j) = 1 + log(2\pi) + log|det(\Sigma)| \tag{4}$$

where Σ is the 2-by-2 covariance matrix of F_i and F_j. The determinant of Σ is

$$det(\Sigma) = \sigma_i^2\sigma_j^2(1 - \rho_{ij}^2) \tag{5}$$

where ρ_{ij} is the correlation coefficient of F_i and F_j

$$\rho_{ij} = E[(F_i - E[F_i])(F_j - E[F_j])] / (\sigma_i^2 \sigma_j^2) \quad (6)$$

Using the formula of Eq.(5), we get

$$h(F_i, F_j) = 1 + log(2\pi) + log[\sigma_i^2 \sigma_j^2 (1 - \rho_{ij}^2)] / 2 \quad (7)$$

By substituting Eqs.(2), (3), and (7) in (1), we get

$$I(F_i ; F_j) = - log(1 - \rho_{ij}^2) / 2 \quad (8)$$

From Eq.(8), we may make the following statements:

1. If F_i and F_j are uncorrelated, the correlation coefficient ρ_{ij} is reduced to zero, and the mutual information $I(F_i ; F_j)$ becomes very small.
2. If F_i and F_j are highly positively correlated, the correlation coefficient ρ_{ij} is close to 1, and mutual information $I(F_i ; F_j)$ becomes very large.

Both theoretical and experimental results (Clemen et al., 1985) have indicated that when individual networks in an ensemble are unbiased, average procedures are most effective in combining them when errors in the individual networks are negatively correlated and moderately effective when the errors are uncorrelated. There is little to be gained from average procedures when the errors are positively correlated. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of the population should be minimised. Minimising the mutual information between each individual neural network and the rest of the population is equivalent to minimising the correlation coefficient between them.

Negative Correlation Learning

Given the training data set $D = \{(\mathbf{x}(1),\mathbf{y}(1)), \ldots, (\mathbf{x}(N),\mathbf{y}(N))\}$, we consider estimating $\mathbf{y}$ by forming a neural network ensemble whose output is a simple averaging of outputs F_i of a set of neural networks. All the individual networks in the ensemble are trained on the same training data set D

$$F(n) = \frac{1}{M} \sum_{i=1}^{M} F_i(n) \quad (9)$$

where $F_i(n)$ is the output of individual network i on the nth training pattern $\mathbf{x}(n)$, $F(n)$ is the output of the neural network ensemble on the nth training pattern, and M is the number of individual networks in the neural network ensemble.

The idea of negative correlation learning is to introduce a correlation penalty term into the error function of each individual network so that the individual network can be trained simultaneously and interactively. The error function E_i for individual i on the training data set D in negative correlation learning is defined by

$$E_i = \frac{1}{N} \sum_{n=1}^{N} \left[\frac{1}{2} (F_i(n) - y(n))^2 + \lambda p_i(n) \right] \tag{10}$$

where N is the number of training patterns, $E_i(n)$ is the value of the error function of network i at presentation of the nth training pattern and $y(n)$ is the desired output of the nth training pattern. The first term in the right side of Eq.(10) is the mean-squared error of individual network i. The second term p_i is a correlation penalty function. The purpose of minimising p_i is to negatively correlate each individual's error with errors for the rest of the ensemble. The parameter λ is used to adjust the strength of the penalty.

The penalty function p_i has the form

$$p_i(n) = -(F_i(n)-F(n))^2 / 2 \tag{11}$$

The partial derivative of E_i with respect to the output of individual i on the nth training pattern is

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F_i(n) - y(n) - \lambda(F_i(n) - F(n)) \tag{12}$$

where we have made use of the assumption that the output of ensemble $F(n)$ has constant value with respect to $F_i(n)$. The value of parameter λ lies inside the range $0 \le \lambda \le 1$ so that both $(1-\lambda)$ and λ have nonnegative values. BP (Rumelhart et al., 1996) algorithm has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eq.(12) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an *epoch*. Negative correlation learning from Eq.(12) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda(F_i(n) - F(n))$ for the ith neural network.

From Eqs.(10), (11) and (12), we may make the following observations:

1. During the training process, all the individual networks interact with each other through their penalty terms in the error functions. Each network F_i minimises not only the difference between $F_i(n)$ and $y(n)$, but also the difference between $F(n)$ and $y(n)$. That is, negative correlation learning considers errors what all other neural networks have learned while training a neural network.
2. For $\lambda=0.0$, there are no correlation penalty terms in the error functions of the individual networks, and the individual networks are just trained independently using BP. That is, independent training using BP for the individual networks is a special case of negative correlation learning.
3. For $\lambda=1$, from Eq.(12) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - y(n) \tag{13}$$

Note that the error of the ensemble for the nth training pattern is defined by

$$E_{ensemble} = \frac{1}{2}\left(\frac{1}{M}\sum_{i=1}^{M} F_i(n) - y(n)\right)^2 \tag{14}$$

The partial derivative of $E_{ensemble}$ with respect to F_i on the nth training pattern is

$$\frac{\partial E_{ensemble}}{\partial F_i(n)} = \frac{1}{M}(F(n) - y(n)) \tag{15}$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ensemble}}{\partial F_i(n)} \tag{16}$$

The minimisation of the error function of the ensemble is achieved by minimising the error functions of the individual networks. From this point of view, negative correlation learning provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual networks.

ANALYSIS BASED ON MEASURING MUTUAL INFORMATION

In order to understand why and how negative correlation learning works, this section analyses it through measuring mutual information on a regression task in three cases: noise-free condition, small noise condition and large noise condition.

Simulation Setup

The regression function investigated here is

$$f(x) = \frac{1}{13}[10\sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5)] - 1 \quad (17)$$

where $\mathbf{x}=[x_1, ..., x_5]$ is an input vector whose components lie between zero and one. The value of $f(\mathbf{x})$ lies in the interval [-1, 1]. This regression task has been used by Jacobs (1997) to estimate the bias of mixture-of-experts architectures and the variance and covariance of experts' weighted outputs.

Twenty-five training sets, $(x^{(k)}(l), y^{(k)}(l))$, $l=1, ..., L$, $L=500$, $k=1, ..., K$, $K=25$, were created at random. Each set consisted of 500 input-output patterns in which the components of the input vectors were independently sampled from a uniform distribution over the interval (0, 1). In the noise-free condition, the target outputs were not corrupted by noise; in the small noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2=0.1$ to the function $f(\mathbf{x})$; in the large noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2=0.2$ to the function $f(\mathbf{x})$. A testing set of 1,024 input-output patterns, $(\mathbf{t}(n), d(n))$, $n = 1, ..., N$, $N = 1024$, was also generated. For this set, the components of the input vectors were independently sampled from a uniform distribution over the interval (0, 1), and the target outputs were not corrupted by noise in all three conditions. Each individual network in the ensemble is a multi-layer perceptron with one hidden layer. All the individual networks have 5 hidden nodes in an ensemble architecture. The hidden node function is defined by the logistic function

$$\varphi(y) = \frac{1}{1+\exp(-y)} \quad (18)$$

The network output is a linear combination of the outputs of the hidden nodes.

For each estimation of mutual information among an ensemble, 25 simulations were conducted. In each simulation, the ensemble was trained on a different training set from the same initial weights distributed inside a small range so that different simulations of an ensemble yielded different performances solely due to the use of different training sets. Such simulation setup follows the suggestions from Jacobs (1997).

Measurement of Mutual Information

The average outputs of the ensemble and the individual network i on the nth pattern in the testing set, (t(n), d(n)), n = 1, ..., N, are denoted and given respectively by

$$\overline{F}(\mathrm{t}(n)) = \frac{1}{K}\sum_{k=1}^{K} F^{(k)}(\mathrm{t}(n)) \tag{19}$$

and

$$\overline{F}_i(\mathrm{t}(n)) = \frac{1}{K}\sum_{k=1}^{K} F_i^{(k)}(\mathrm{t}(n)) \tag{20}$$

where $F^{(k)}(t(n))$ and $F_i^{(k)}(t(n))$ are the outputs of the ensemble and the individual network i on the nth pattern in the testing set from the kth simulation, respectively, and $K=25$ is the number of simulations. From Eq.(6), the correlation coefficient between network i and network j is given by

$$\rho_{ij} = \frac{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_i^{(k)}(\mathrm{t}(n)) - \overline{F}_i(\mathrm{t}(n))\right)\left(F_j^{(k)}(\mathrm{t}(n)) - \overline{F}_j(\mathrm{t}(n))\right)}{\sqrt{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_i^{(k)}(\mathrm{t}(n)) - \overline{F}_i(\mathrm{t}(n))\right)^2 \sum_{n=1}^{N}\sum_{k=1}^{K}\left(F_j^{(k)}(\mathrm{t}(n)) - \overline{F}_j(\mathrm{t}(n))\right)^2}} \tag{21}$$

From Eq.(8), the integrated mutual information among the ensembles can be defined by

$$E_{mi} = -\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1, j\neq i}^{M} log(1-\rho_{ij}^2) \tag{22}$$

We may also define the integrated mean-squared error (MSE) on the testing set as

$$E_{mse} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{K}\sum_{k=1}^{K}(F^{(k)}(\mathrm{t}(n)) - d(n))^2 \quad (23)$$

The integrated mean-squared error E_{train_mse} on the training set is given by

$$E_{train\_mis} = \frac{1}{L}\sum_{l=1}^{L}\frac{1}{K}\sum_{k=1}^{K}(F^{(k)}(\mathrm{x}^{(k)}(l)) - y^{(k)}(l))^2 \quad (24)$$

Results in the Noise-Free Condition

The results of negative correlation learning in the noise-free condition for the different values of λ at epoch 2000 are given in Table 1. The results suggest that both E_{train_mse} and E_{test_mse} appeared to decrease with the increasing value of λ. The mutual information E_{mi} among the ensemble decreased as the value of λ increased when $0 \le \lambda \le 0.5$. However, when λ increased further to 0.75 and 1, the mutual information E_{mi} had larger values. The reason of having larger mutual information at $\lambda = 0.75$ and $\lambda = 1$ is that some correlation coefficients had negative values and the mutual information depends on the absolute values of correlation coefficients.

In order to find out why E_{train_mse} decreased with increasing value of λ, the concept of capability of a trained ensemble is introduced. The capability of a trained ensemble is measured by its ability of producing correct input-output mapping on the training set used, specifically, by its integrated mean-squared error E_{train_mse} on the training set. The smaller E_{train_mse} is, the larger capability the trained ensemble has.

Results in the Noise Conditions

Table 2 and Table 3 compare the performance of negative correlation learning for different strength parameters in both small noise (variance $\sigma^2 = 0.1$) and large

Table 1: The results of negative correlation learning in the noise-free condition for different l values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	0.3706	0.1478	0.1038	0.1704	0.6308
$E_{test\ mse}$	0.0016	0.0013	0.0011	0.0007	0.0002
$E_{train\ mse}$	0.0013	0.0010	0.0008	0.0005	0.0001

noise (variance $\sigma^2 = 0.2$) conditions. The results show that there were same trends for E_{mi}, E_{test_mse} and E_{train_mse} in both noise-free and noise conditions when $\lambda \leq 0.5$. That is, E_{mi}, E_{test_mse} and E_{train_mse} appeared to decrease with the increasing value of λ. However, E_{test_mse} appeared to decrease first and then increase with the increasing value of λ.

In order to find out why E_{test_mse} showed different trends in noise-free and noise conditions when $\lambda = 0.75$ and $\lambda = 1$, the integrated mean-squared error E_{train_mse} on the training set was also shown in Tables 1, 2 and 3. When $\lambda = 0$, the neural network ensemble trained had relatively large E_{train_mse}. It indicated that the capability of the neural network ensemble trained was not big enough to produce correct input-output mapping (i.e., it was underfitting) for this regression task. When $\lambda = 1$, the neural network ensemble trained learned too many specific input-output relations (i.e., it was overfitting), and it might memorise the training data and therefore be less able to generalise between similar input-output patterns. Although the overfitting was not observed for the neural network ensemble used in the noise-free condition, too large capability of the neural network ensemble will lead to overfitting for both noise-free and noise conditions because of the ill-posedness of any finite training set (Friedman, 1994).

Choosing a proper value of λ is important, and also problem dependent. For the noise conditions used for this regression task and the ensemble architectured used, the performance of the ensemble was optimal for $\lambda = 0.5$ among the tested values of λ in the sense of minimising the MSE on the testing set.

Table 2: The results of negative correlation learning in the small noise condition for different λ values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	6.5495	3.8761	1.4547	0.3877	0.2431
E_{test_mse}	0.0137	0.0128	0.0124	0.0126	0.0290
E_{train_mse}	0.0962	0.0940	0.0915	0.0873	0.0778

Table 3: The results of negative correlation learning in the large noise condition for different λ values at epoch 2000

λ	0	0.25	0.5	0.75	1
E_{mi}	6.7503	3.9652	1.6957	0.4341	0.2030
E_{test_mse}	0.0249	0.0235	0.0228	0.0248	0.0633
E_{train_mse}	0.1895	0.1863	0.1813	0.1721	0.1512

ANALYSIS BASED ON DECISION BOUNDARIES

This section analyses the decision boundaries constructed by both negative correlation learning and the independent training. The independent training is a special case of negative correlation learning for $\lambda = 0.0$ in Eq.(12).

Simulation Setup

The objective of the pattern classification problem is to distinguish between two classes of overlapping, two-dimensional, Gaussian-distributed patterns labeled 1 and 2. Let Class 1 and Class 2 denote the set of events for which a random vector **x** belongs to patters 1 and 2, respectively. We may then express the conditional probability density functions for the two classes:

$$f_X(x) = \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{1}{2\sigma_1^2}||x - \mu_1||^2\right) \tag{25}$$

where mean vector $\mu_1 = [0,0]^T$ and variance $\sigma_1^2 = 1$.

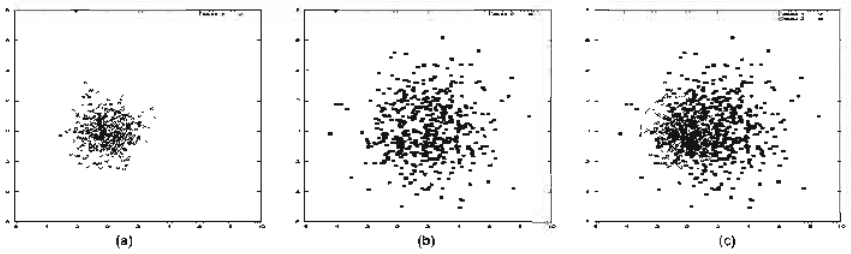
$$f_X(x) = \frac{1}{2\pi\sigma_2^2} \exp\left(-\frac{1}{2\sigma_2^2}||x - \mu_2||^2\right) \tag{26}$$

where mean vector $\mu_2 = [0,0]^T$ and variance $\sigma_2^2 = 4$. The two classes are assumed to be equiprobable; that is $p_1 = p_2 = ½$. The costs for misclassifications are assumed to be equal, and the costs for correct classifications are assumed to be zero. On this basis, the (optimum) Bayes classifier achieves a probability of correct classification $p_c = 81.51$ percent. The boundary of the Bayes classifier consists of a circle of center $[-2/3, 0]^T$ and radius $r = 2.34$; 1000 points from each of two processes were generated for the training set. The testing set consists of 16,000 points from each of two classes.

Figure 1 shows individual scatter diagrams for classes and the joint scatter diagram representing the superposition of scatter plots of 500 points from each of two processes. This latter diagram clearly shows that the two distributions overlap each other significantly, indicating that there is inevitably a significant probability of misclassification.

The ensemble architecture used in the experiments has three networks. Each individual network in the ensemble is a multi-layer perceptron with one hidden layer. All the individual networks have three hidden nodes in an ensemble architecture.

Figure 1: (a) Scatter plot of Class 1; (b) Scatter plot of Class 2; (c) Combined scatter plot of both classes; the circle represents the optimum Bayes solution



Both hidden node function and output node function are defined by the logistic function in Eq.(18).

Experimental Results

The results presented in Table 4 pertain to 10 different runs of the experiment, with each run involving the use of 2,000 data points for training and 32,000 for testing. Figures 2 and 3 compare the decision boundaries constructed by negative

Figure 2: Decision boundaries formed by the different networks trained by the negative correlation learning ($\lambda = 0.75$): (a) Network 1; (b) Network 2; (c) Network 3; (d) Ensemble; the circle represents the optimum Bayes solution

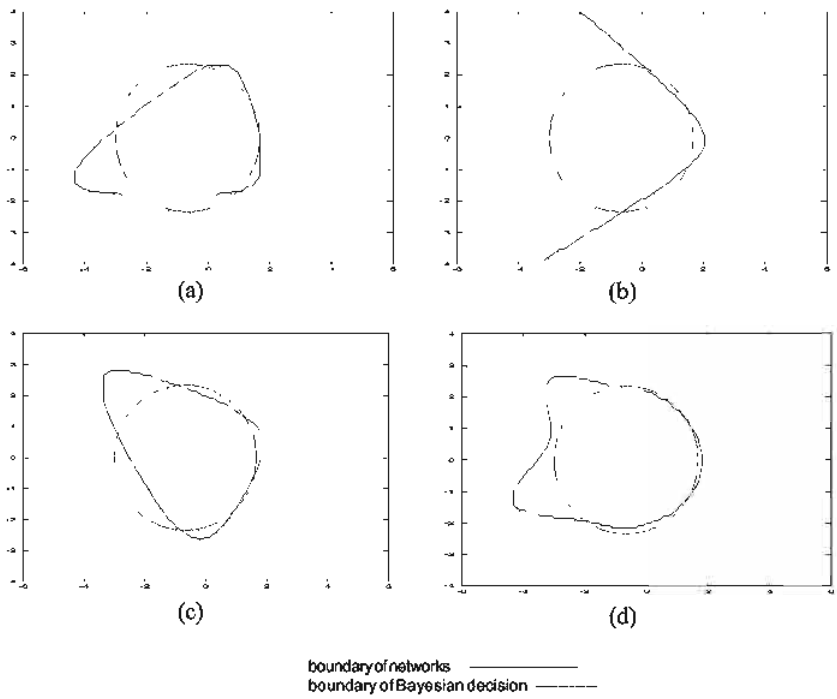
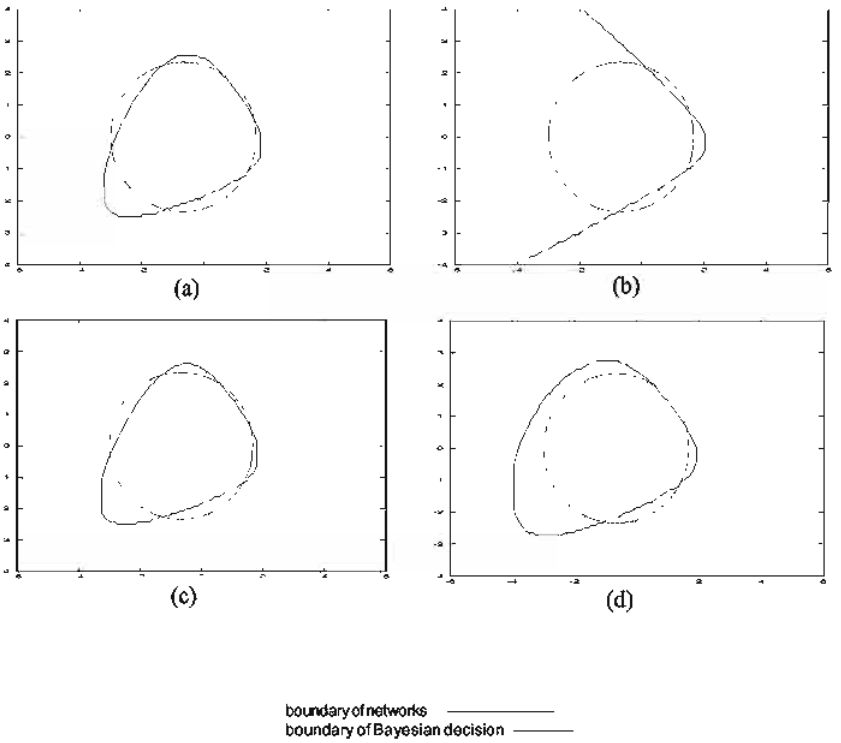


Figure 3: Decision boundaries formed by the different networks trained by the independent training (i.e., $\lambda = 0.0$ in negative correlation learning): (a) Network 1; (b) Network 2; (c) Network 3; (d) Ensemble; the circle represents the optimum Bayes solution



correlation learning and the independent training. In comparison of the average correct classification percentage and the decision boundaries obtained by the two ensemble learning methods, it is clear that negative correlation learning outperformed the independent training method. Although the classification performance of individual networks in the independent training is relatively good, the overall performance of the entire ensemble was not improved because different networks, such as Network 1 and Network 3 in Figure 3, tended to generate the similar decision boundaries.

The percentage of correct classification of the ensemble trained by negative correlation is 81.41, which is almost equal to that realised by the Bayesian classifier. Figure 2 clearly demonstrates that negative correlation learning is capable of constructing a decision between Class 1 and Class 2 that is almost as good as the optimum decision boundary. It is evident from Figure 2 that different individual networks trained by negative correlation learning were able to specialise to different parts of the testing set.

Table 4: Comparison between negativecorrelation learning (NCL) ($\lambda = 0.75$) and the independent training (i.e., $\lambda = 0.0$ in negative correlation learning) on the classification performance of individual networks and the ensemble; the results are the average correct classification percentage on the testing set over 10 independent runs

Methods	Net 1	Net 2	Net 3	Ensemble
NCL	81.11	75.26	73.09	81.03
Independent Training	81.13	80.49	81.13	80.99

ANALYSIS BASED ON THE CORRECT RESPONSE SETS

In this section, negative correlation learning was tested on the Australian credit card assessment problem. The problem is how to assess applications for credit cards based on a number of attributes. There are 690 patterns in total. The output has two classes. The 14 attributes include 6 numeric values and 8 discrete ones, the latter having from 2 to 14 possible values. The Australian credit card assessment problem is a classification problem which is different from the regression type of tasks, whose outputs are continuous. The data set was obtained from the UCI machine learning benchmark repository. It is available by anonymous ftp at ics.uci.edu (128.195.1.1) in directory /pub/machine-learning-databases.

Experimental Setup

The data set was partitioned into two sets: a training set and a testing set. The first 518 examples were used for the training set, and the remaining 172 examples for the testing set. The input attributes were rescaled to between 0.0 and 1.0 by a linear function. The output attributes of all the problems were encoded using a *1-of-m* output representation for m classes. The output with the highest activation designated the class. The aim of this section is to study the difference between negative correlation learning and independent training, rather than to compare negative correlation learning with previous work. The experiments used such a single train-and-test partition.

The ensemble architecture used in the experiments has 4 networks. Each individual network is a feedforward network with one hidden layer. Both hidden node function and output node function are defined by the logistic function in Eq.(18). All the individual networks have 10 hidden nodes. The number of training

epochs was set to 250. The strength parameter λ was set to 1.0. These parameters were chosen after limited preliminary experiments. They are not meant to be optimal.

Experimental Results

Table 5 shows the average results of negative correlation learning over 25 runs. Each run of negative correlation learning was from different initial weights. The ensemble with the same initial weight setup was also trained using BP without the correlation penalty terms (i.e., $\lambda = 0.0$ in negative correlation learning). Results are also shown in Table 5. For this problem, the simple averaging defined in Eq.(9) was first applied to decide the output of the ensemble. For the simple averaging, it was surprising that the results of negative correlation learning with $\lambda = 1.0$ were similar to those of independent training. This phenomenon seems contradictory to the claim that the effect of the correlation penalty term is to encourage different individual networks in an ensemble to learn different parts or aspects of the training data. In order to verify and quantify this claim, we compared the outputs of the individual networks trained with the correlation penalty terms to those of the individual networks trained without the correlation penalty terms.

Table 5: Comparison of error rates between negative correlation learning ($\lambda = 1.0$) and independent training (i.e., $\lambda = 0.0$ in negative correlation learning) on the Australian credit card assessment problem; the results were averaged over 25 runs. "Simple Averaging" and "Winner-Takes-All" indicate two different combination methods used in negative correlation learning, Mean, SD, Min and Max indicate the mean value,standard deviation, minimum and maximum value, respectively

	Error Rate	Simple Averageing	Winner-Takes-All
$\lambda = 1.0$	Mean	0.1337	0.1195
	SD	0.0068	0.0052
	Min	0.1163	0.1105
	Max	0.1454	0.1279
$\lambda = 0.0$	Mean	0.1368	0.1384
	SD	0.0048	0.0049
	Min	0.1279	0.1279
	Max	0.1454	0.1512

Table 6: The sizes of the correct response sets of individual networks created respectively by negative correlation learning (λ = 1.0) and independent training (i.e., λ = 0.0 in negative correlation learning) on the testing set and the sizes of their intersections for the Australian credit card assessment problem; the results were obtained from the first run among the 25 runs

$\lambda = 1.0$			$\lambda = 0.0$		
$\Omega_1 = 147$	$\Omega_2 = 143$	$\Omega_2 = 138$	$\Omega_1 = 149$	$\Omega_2 = 147$	$\Omega_2 = 148$
$\Omega_4 = 143$	$\Omega_{12} = 138$	$\Omega_{13} = 124$	$\Omega_4 = 148$	$\Omega_{12} = 147$	$\Omega_{13} = 147$
$\Omega_{14} = 141$	$\Omega_{23} = 116$	$\Omega_{24} = 133$	$\Omega_{14} = 147$	$\Omega_{23} = 147$	$\Omega_{24} = 146$
$\Omega_{34} = 123$	$\Omega_{123} = 115$	$\Omega_{124} = 133$	$\Omega_{34} = 146$	$\Omega_{123} = 147$	$\Omega_{124} = 146$
$\Omega_{134} = 121$	$\Omega_{234} = 113$	$\Omega_{1234} = 113$	$\Omega_{134} = 146$	$\Omega_{234} = 146$	$\Omega_{1234} = 146$

Two notions were introduced to analyse negative correlation learning. They are the correct response sets of individual networks and their intersections. The correct response set S_i of individual network *i* on the testing set consists of all the patterns in the testing set which are classified correctly by the individual network *i*. Let Ω_i denote the size of set S_i, and $\Omega_{i1i2\cdots ik}$ denote the size of set $S_{i1} \cap S_{i2} \cap \cdots \cap S_{ik}$. Table 6 shows the sizes of the correct response sets of individual networks and their intersections on the testing set, where the individual networks were respectively created by negative correlation learning and independent training. It is evident from Table 6 that different individual networks created by negative correlation learning were able to specialise to different parts of the testing set. For instance, in Table 6 the sizes of both correct response sets S_2 and S_4 at $\lambda = 1.0$ were 143, but the size of their intersection $S_2 \cap S_4$ was 133. The size of $S_1 \cap S_2 \cap S_3 \cap S_4$ was only 113. In contrast, the individual networks in the ensemble created by independent training were quite similar. The sizes of correct response sets S_1, S_2, S_3 and S_4 at $\lambda = 0.0$ were from 147 to 149, while the size of their intersection set $S_1 \cap S_2 \cap S_3 \cap S_4$ reached 146. There were only three different patterns correctly classified by the four individual networks in the ensemble.

In simple averaging, all the individual networks have the same combination weights and are treated equally. However, not all the networks are equally important. Because different individual networks created by negative correlation learning were able to specialise to different parts of the testing set, only the outputs of these specialists should be considered to make the final decision about the ensemble for this part of the testing set. In this experiment, a winner-takes-all method was applied to select such networks. For each pattern of the testing set,

the output of the ensemble was only decided by the network whose output had the highest activation. Table 5 shows the average results of negative correlation learning over 25 runs using the winner-takes-all combination method. The winner-takes-all combination method improved negative correlation learning significantly because there were good and poor networks for each pattern in the testing set, and winner-takes-all selected the best one. However, it did not improve the independent training much because the individual networks created by the independent training were all similar to each other.

CONCLUSIONS

This chapter describes negative correlation learning for designing neural network ensembles. It can be regarded as one way of decomposing a large problem into smaller and specialised ones, so that each sub-problem can be dealt with by an individual neural network relatively easily. A correlation penalty term in the error function was proposed to minimise mutual information and encourage the formation of specialists in the ensemble.

Negative correlation learning has been analysed in terms of mutual information on a regression task in the different noise conditions. Unlike independent training which creates larger mutual information among the ensemble, negative correlation learning can produce smaller mutual information among the ensemble. Through minimisation of mutual information, very competitive results have been produced by negative correlation learning in comparison with independent training.

This chapter compares the decision boundaries and the correct response sets constructed by negative correlation learning and the independent training for two pattern classification problems. The experimental results show that negative correlation learning has a very good classification performance. In fact, the decision boundary formed by negative correlation learning is nearly close to the optimum decision boundary generated by the Bayes classifier.

There are, however, some issues that need resolving. No special considerations were made in optimisation of the size of the ensemble and strength parameter λ in this chapter. Evolutionary ensembles with negative correlation learning for optimisation of the size of the ensemble had been studied on the classification problems (Liu, Yao & Higuchi, 2000).

REFERENCES

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Trans. Appl. Comp.*, AC-19, 716-723.

Clemen, R. T., & Winkler, R. L. (1985). Limits for the precision and value of information from dependent sources. *Operations Research*, 33:427-442.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y. & Vapnik, V. (1994). Boosting and other ensemble methods. *Neural Computation*, 6:1289-1301.

Drucker, H., Schapire, R. & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In Hanson, S. J., Cowan, J. D. & Giles,C. L. (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 42-49. San Mateo, CA: Morgan Kaufmann.

Friedman, J. H. (1994). An overview of predictive learning and function approximation. In V. Cherkassky, J. H. Friedman, and H. Wechsler, (Eds.), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pp. 1-61. Springer-Verlag, Heidelberg, Germany.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(10):993-1001.

Jacobs, R. A. (1997). Bias/variance analyses of mixture-of-experts architectures. *Neural Computation*, 9:369-383.

Jacobs, R. A. & Jordan, M. I. (1991). A competitive modular connectionist architecture. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, (Eds.), *Advances in Neural Information Processing Systems 3*, pp. 767-773. Morgan Kaufmann, San Mateo, CA.

Jacobs, R. A., Jordan, M. I. & Barto, A. G. (1991). Task decomposition through competition in a modular connectionist architecture: the what and where vision task. *Cognitive Science*, 15:219-250.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J. & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79-87.

Jordan, M. I. & Jacobs, R. A. (1994). Hierarchical mixtures-of-experts and the em algorithm. *Neural Computation*, 6:181-214.

Liu, Y. & Yao, X. (1998a). Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems*, 4:176-185.

Liu, Y. & Yao, X. (1998b). A cooperative ensemble learning system. *In Proceedings of the 1998 IEEE International Joint Conference on Neural Networks (IJCNN'98)*, pages 2202-2207. IEEE Press, Piscataway, NJ, USA.

Liu, Y. & Yao, X. (1999). Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):716-725.

Liu, Y., Yao, X., & Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Trans. on Evolutionary Computation*, 4(4):380-725.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. New York: McGraw Hill.

Opitz, D. W. & Shavlik, J. W. (1996). Actively searching for an effective neural network ensemble. *Connection Science*, 8:337-353.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14:465-471.

Rosen, B. E. (1996). Ensemble learning using decorrelated neural networks. *Connection Science*, 8:373-383.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. I, pp. 318-362. MIT Press, Cambridge, MA.

Sarkar, D. (1996). Randomness in generalization ability: A source to improve it. *IEEE Trans. on Neural Networks*, 7(3):676-685.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:197-227.

Selfridge, O. G. (1958). Pandemonium: a paradigm for learning. *Mechanisation of Thought Processes: Proceedings of a Symp*. Held at the National Physical Lab., pp. 513-526. HMSO, London.

Sharkey, A. J. C. (1996). On combining artificial neural nets. *Connection Science*, 8:299-313.

van der Lubbe, J. C. A. (1997). *Information Theory*. Cambridge: Cambridge University Press.

van der Lubbe, J. C. A. (1999). *Information Theory*. (2nd ed) Prentice-Hall International, Inc.

Wallace, C. S., & Patrick, J. D. (1991). *Coding Decision Trees*. Technical Report 91/153, Department of Computer Science, Monash University, Clayton, Victoria 3168, Australia, August.

Wolpert, D. H. (1990). A mathematical theory of generalization. *Complex Systems*, 4:151–249.

Chapter II

A Perturbation Size-Independent Analysis of Robustness in Neural Networks by Randomized Algorithms

C. Alippi
Politecnico di Milano, Italy

ABSTRACT

This chapter presents a general methodology for evaluating the loss in performance of a generic neural network once its weights are affected by perturbations. Since weights represent the "knowledge space" of the neural model, the robustness analysis can be used to study the weights/performance relationship. The perturbation analysis, which is closely related to sensitivity issues, relaxes all assumptions made in the related literature, such as the small perturbation hypothesis, specific requirements on the distribution of perturbations and neural variables, the number of hidden units and a given neural structure. The methodology, based on Randomized Algorithms, allows reformulating the computationally intractable problem of robustness/sensitivity analysis in a probabilistic framework characterised by a polynomial time solution in the accuracy and confidence degrees.

INTRODUCTION

The evaluation of the effects induced by perturbations affecting a neural computation is relevant from the theoretical point of view and in developing an embedded device dedicated to a specific application.

In the first case, the interest is in obtaining a reliable and easy to be generated measure of the performance loss induced by perturbations affecting the weights of a neural network. The relevance of the analysis is obvious since weights characterise the "knowledge space" of the neural model and, hence, its inner nature. In this direction, a study of the evolution of the network's weights over training time allows for understanding the mechanism behind the generation of the knowledge space. Conversely, the analysis of a specific knowledge space (fixed configuration for weights) provides hints about the relationship between the weights space and the performance function. The latter aspect is of primary interest in recurrent neural networks where even small modifications of the weight values are critical to performance (e.g., think of the stability of an intelligent controller comprising a neural network and issues leading to robust control).

The second case is somehow strictly related to the first one and covers the situation where the neural network must be implemented in a physical device. The optimally trained neural network becomes the "golden unit" to be implemented within a finite precision representation environment as it happens in mission-critical applications and embedded systems. In these applications, behavioural perturbations affecting the weights of a neural network abstract uncertainties associated with the implementation process, such as finite precision representations (e.g., truncation or rounding in a digital hardware, fixed or low resolution floating point representations), fluctuations of the parameters representing the weights in analog solutions (e.g., associated with the production process of a physical component), ageing effects, or more complex and subtle uncertainties in mixed implementations.

The sensitivity/robustness issue has been widely addressed in the neural network community with a particular focus on specific neural topologies.

More in detail, when the neural network is composed of linear units, the analysis is straightforward and the relationship between perturbations and the induced performance loss can be obtained in a closed form (Alippi & Briozzo, 1998). Conversely, when the neural topology is non-linear, which is mostly the case, several authors assume the small perturbation hypothesis or particular hypothesis about the stochastic nature of the neural computation. In both cases, the assumptions make the mathematics more amenable with the positive consequence that a relationship between perturbations and performance loss can be derived (e.g., see Alippi & Briozzo, 1998; Pichè, 1995). Unfortunately, these analyses introduce hypotheses which are not always satisfied in all real applications.

Another classic approach requires expanding with Taylor the neural computation around the nominal value of the trained weights. A subsequent linear arised analysis follows which allows for solving the sensitivity issue (e.g., Pichè, 1995). Anyway, the validity of such approaches depend, in turn, on the validity of the small perturbation hypothesis: how to understand a priori if a perturbation is small for a given application?

In other applications the small perturbation hypothesis cannot be accepted being the involved perturbations everything but small. As an example we have the development of a digital embedded system. There, the designer has to reduce as possible the dimension of the weights by saving bits; this produces a positive impact on cost, memory size and power consumption of the final device.

Differently, other authors avoid the small perturbation assumption by focusing the attention on very specific neural network topologies and/or introducing particular assumptions regarding the distribution of perturbations, internal neural variables and inputs (Stevenson, Winter & Widrow, 1990; Alippi, Piuri & Sami, 1995).

Other authors have considered the sensitivity analysis under the small perturbation hypothesis to deal with implementation aspects. In this case, perturbations are specifically related to finite precision representations of the interim variables characterising the neural computation (Holt & Hwang, 1993; Dundar & Rose, 1995).

Differently from the limiting approaches provided in the literature, this chapter suggests a robustness/sensitivity analysis in the large, i.e., without assuming constraints on the size or nature of the perturbation; as such, small perturbation situations become only a subcase of the theory. The analysis is general and can be applied to all neural topologies, both static and recurrent in order to quantify the performance loss of the neural model when perturbations affect the model's weights.

The suggested sensitivity/robustness analysis can be applied to *All* neural network models involved in system identification, control signal/image processing and automation-based applications without any restriction. In particular, the analysis allows for solving the following problems:

- Quantify the robustness of a generically trained neural network by means of a suitable, easily to be computed and reliable robustness index;
- Compare different neural networks, solving a given application by ranking them according to their robustness;
- Investigate the criticality of a recurrent model ("stability" issue) by means of its robustness index;

- Study the efficacy and effectiveness of techniques developed to improve the robustness degree of a neural network by inspecting the improvement in robustness.

The key elements of the perturbation analysis are Randomised Algorithms – RAs- (Vidyasagar, 1996, 1998; Tempo & Dabbene, 1999; Alippi, 2002), which transform the computationally intractable problem of evaluating the robustness of a generic neural network with respect to generic, continuous perturbations, in a tractable problem solvable with a polynomial time algorithm by resorting to probability.

The increasing interest and the extensive use of Randomised Algorithms in control theory, and in particular in the robust control area (Djavan, Tulleken, Voetter, Verbruggen & Olsder, 1989; Battarcharyya, Chapellat & Keel, 1995; Bai & Tempo, 1997; Chen & Zhou, 1997; Vidyasagar, 1998; Tempo & Dabbene, 1999, Calafiore, Dabbene & Tempo, 1999), make this versatile technique extremely interesting also for the neural network researcher.

We suggest the interested reader to refer to Vidyasagar (1998) and Tempo and Dabbene (1999) for a deep analysis of the use of RAs in control applications; the author forecasts an increasing use of Randomised Algorithms in the analysis and synthesis of intelligent controllers in the neural network community.

The structure of the chapter is as follows. We first formalise the concept of robustness by identifying a natural and general index for robustness. Randomised Algorithms are then briefly introduced to provide a comprehensive analysis and adapted to estimate the robustness index. Experiments then follow to shed light on the use of the theory in identifying the robustness index for static and recurrent neural models.

A GENERAL ROBUSTNESS/SENSITIVITY ANALYSIS FOR NEURAL NETWORKS

In the following we consider a *generic* neural network implementing the $\hat{y} = f(\hat{\theta}, x)$ function where $\hat{\theta}$ is the weight (and biases) vector containing all the trained free parameters of the neural model.

In several neural models, and in particular in those related to system identification and control, the relationship between the inputs and the output of the system are captured by considering a regressor vector φ, which contains a limited time-window of actual and past inputs, outputs and possibly predicted outputs.

Of particular interest, in the zoo of neural models, are those which can be represented by means of the model structures $\hat{y}(t) = f(\varphi)$ where function $f(\cdot)$ is a regression-type neural network, characterised by N_φ inputs, N_h non-linear hidden units and a single effective linear/non-linear output (Ljung, 1987; Hertz, Krog & Palmer, 1991; Hassoun, 1995; Ljung, Sjoberg & Hjalmarsson, 1996).

The absence/presence of a dynamic in the system can be modelled by a suitable number of delay elements (or time lags), which may affect inputs (time history on external inputs u) system outputs (time history on $y(t)$) on predicted outputs (time history on $\hat{y}(t)$) or residuals (time history on $e(t) = \hat{y}(t) - y(t)$). Where it is needed $y(t)$, $\hat{y}(t)$ and $e(t)$ are vectorial entities, a component for each independent distinct variable.

Several neural model structures have been suggested in the literature, which basically differ in the regressor vector. Examples are, NARMAX and NOE topologies. NARMAX structure can be obtained by considering both past inputs and outputs of the system to infer $y(t)$. We have:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u), y(t-1), \cdots, y(t-n_y), \ldots, e(t-1), \cdots, e(t-n_e)\right]$$

Differently, the NOE structure processes only past inputs and predicted outputs, i.e.:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u), \hat{y}(t-1), \cdots, \hat{y}(t-n_y)\right].$$

Static neural networks, such as classifiers, can be obtained by simply considering external inputs:

$$\varphi = \left[u(t), u(t-1), \cdots, u(t-n_u)\right].$$

Of course, different neural models can be considered, e.g., fully recurrent and well fit with the suggested robustness analysis.

A general, perturbation size independent, model-independent robustness analysis requires the evaluation of the loss in performance induced by a generic perturbation, in our analysis affecting the weights of a generic neural network. We denote by $y_\Delta(x) = f_\Delta(\theta, \Delta, x)$ the mathematical description of the perturbed computation and by $\Delta \in D \subseteq \Re^p$ a generic p-dimensional perturbation vector, a component for each independent perturbation affecting the neural computation $\hat{y}(t)$. The perturbation space D is characterised in stochastic terms by providing the probability density function pdf_D.

To measure the discrepancy between $y_\Delta(x)$ and $y(t)$ or $\hat{y}(t)$, we consider a generic loss function $U(\Delta)$. In the following we only assume that such performance loss function is measurable according to Lebesgue with respect to D. Lebesgue measurability for $U(\Delta)$ allows us for taking into account an extremely large class of loss functions.

Common examples for U are the Mean Square Error—MSE—loss functions

$$U(\Delta)=\frac{1}{N_x}\sum_{i=1}^{N_x}(\hat{y}(x_i)-\hat{y}(x_i,\Delta))^2 \text{ and } U(\Delta)=\frac{1}{N_x}\sum_{i=1}^{N_x}(y(x_i)-\hat{y}(x_i,\Delta))^2 . \quad (1)$$

More specifically, (1)-left compares the perturbed network with $\hat{y}$, which is supposed to be the "golden" error-free unit while (1)-right estimates the performance of the error-affected (perturbed) neural network (generalisation ability of the perturbed neural model).

The formalisation of the impact of perturbation on the performance function can be simply derived:

Definition: Robustness Index

We say that a neural network is robust at level $\bar{\gamma}$ in D, when the robustness index $\bar{\gamma}$ is the minimum positive value for which

$$U(\Delta) \le \bar{\gamma}, \forall \Delta \in D, \ \forall \gamma \ge \bar{\gamma} \quad (2)$$

Immediately, from the definition of robustness index we have that a generic neural network NN_1 is more robust than NN_2 if $\bar{\gamma}_1 < \bar{\gamma}_2$ and the property holds independently from the topology of the two neural networks.

The main problem related to the determination of the robustness index $\bar{\gamma}$ is that we have to compute $U(\Delta), \forall \Delta \in D$ if we wish a tight bound. The $\bar{\gamma}$ -identification problem is therefore intractable from a computational point of view if we relax all assumptions made in the literature as we do.

To deal with the computational aspect we associate a dual probabilistic problem to (2):

Robustness Index: Dual Problem We say that a neural network is robust at level $\bar{\gamma}$ in D with confidence η, when $\bar{\gamma}$ is the minimum positive value for which

$$\Pr(U(\Delta) \le \bar{\gamma}) \ge \eta \quad \text{holds} \quad \forall \Delta \in D, \ \forall \gamma \ge \bar{\gamma} \quad (3)$$

The probabilistic problem is weaker than the deterministic one since it tolerates the existence of a set of perturbations (whose measure according to Lebesgue is 1-η) for which $u(\Delta) > \bar{\gamma}$. In other words, not more than 100η % of perturbations $\Delta \in D$ will generate a loss in performance larger than $\bar{\gamma}$.

Probabilistic and deterministic problems are "close" to each other when we choose, as we do, $\eta = 1$. Note that $\bar{\gamma}$ depends only on the size of D and the neural network structure.

The non-linearity with respect to Δ and the lack of a priori assumptions regarding the neural network do not allow computing (2) in a closed form for the general perturbation case. The analysis, which would imply testing $U\Delta$ in correspondence with a continuous perturbation space, can be solved by resorting to probability according to the dual problem and by applying Randomised Algorithms to solve the robustness/sensitivity problem.

RANDOMIZED ALGORITHMS AND PERTURBATION ANALYSIS

In this paragraph we briefly review the theory behind Randomised Algorithms and adapt them to the robustness analysis problem.

In the following we denote by $p_\gamma = \Pr\{U(\Delta) \le \gamma\}$ the probability that the loss in performance associated with perturbations in D is below a given—but arbitrary—value γ.

Probability p_γ is unknown, cannot be computed in a close form for a generic U function and neural network topology, and its evaluation requires exploration of the whole perturbation space D.

The unknown probability p_γ can be estimated by sampling D with N independent and identically distributed samples Δ_i; extraction must be carried out according to the *pdf* of the perturbation.

For each sample Δ_i we then generate the triplet

$$\{\Delta_i, U(\Delta_i), I(\Delta_i)\}, i = 1, N \text{ where } I(\Delta_i) = \begin{cases} 1 \text{ if } U(\Delta_i) \le \gamma \\ 0 \text{ if } U(\Delta_i) > \gamma \end{cases} \quad (4)$$

The true probability p_γ can now simply be estimated as

$$\hat{p}_N = \frac{1}{N} \sum_{i=1}^{N} I(\Delta_i) \quad (5)$$

Of course, when N tends to infinity, $\hat{p}_N$ converges to p_γ. Conversely, on a finite data set of cardinality N, the discrepancy between $\hat{p}_N$ and p_γ exists and can be

simply measured as $|p_\gamma - \hat{p}_N|$. $|p_\gamma - \hat{p}_N|$ is a random variable which depends on the particular extraction of the N samples since different extractions of N samples from D will provide different estimates for $\hat{p}_N$. By introducing an accuracy degree ε on $|p_\gamma - \hat{p}_N|$ and a confidence level $1-\delta$ (which requests that the $|p_\gamma - \hat{p}_N| \le \varepsilon$ inequality is satisfied at least with probability $1-\delta$), our problem can be formalised by requiring that the inequality

$$\Pr\left\{|p_\gamma - \hat{p}_N| \le \varepsilon\right\} \ge 1 - \delta \tag{6}$$

is satisfied for $\forall \gamma \ge 0$. Of course, we wish to control the accuracy and the confidence degrees of (6) by allowing the user to choose the most appropriate values for the particular need. Finally, by extracting a number of samples from D according to the Chernoff inequality (Chernoff, 1952)

$$N \ge \frac{\ln \frac{2}{\delta}}{2\varepsilon^2} \tag{7}$$

we have that $\Pr\left\{|p_\gamma - \hat{p}_N| \le \varepsilon\right\} \ge 1 - \delta$ holds for $\forall \gamma \ge 0, \forall \delta, \varepsilon \in [0,1]$.

As an example, by considering 5% in accuracy and 99% in confidence, we have to extract 1060 samples from D; with such choice we can approximate p_γ with $\hat{p}_N$ introducing the maximum error 0.05 ($\hat{p}_N - 0.05 \le p_\gamma \le \hat{p}_N + 0.05$) and the inequality holds at least with the probability 0.99.

Other bounds can be considered instead of the Chernoff's one as suggested by Bernoulli and Bienaymè, (e.g., see Tempo & Dabbene, 1999). Nevertheless, the Chernoff's bound improves upon the others and, therefore, should be preferred if we wish to keep minimal the number of samples to be extracted. The Chernoff bound grants that:

- N is independent from the dimension of D (and hence it does not depend on the number of perturbations we are considering in the neural network);
- N is linear in $\ln \frac{1}{\delta}$ and $\frac{1}{\varepsilon^2}$ (hence it is polynomial in the accuracy and confidence degrees).

As a consequence, the dual probabilistic problem related to the identification of the robustness index $\bar{\gamma}$ can be solved with randomised algorithms and therefore with a polynomial complexity in the accuracy and the confidence degrees independently from the number of weights of the neural model network. In fact, by expanding the (6) we have that

$$\Pr\left\{p_\gamma - \hat{p}_N \le \varepsilon\right\} \ge 1-\delta \equiv \Pr\left\{\left|\Pr(u(\Delta)\le\gamma) - \frac{1}{N}\sum_i I(\Delta_i)\right| \le \varepsilon\right\} \ge 1-\delta \quad (8)$$

If accuracy ε and confidence δ are small enough, we can confuse p_γ and $\hat{p}_N$ by committing a small error. As a consequence, the dual probabilistic problem requiring $p_\gamma \ge \eta$ becomes $\hat{p}_N \ge \eta$. We surely assume ε and δ to be small enough in subsequent derivations.

The final algorithm, which allows for testing the robustness degree $\bar{\gamma}$ of a neural network, is:

1. *Select ε and δ sufficiently small to have enough accuracy and confidence.*
2. *Extract from D, according to its pdf, a number of perturbations N as suggested by (7).*
3. *Generate the indicator function $I(\Delta)$ and generate the estimate $\hat{p}_N = \hat{p}_N(\gamma)$ according to (5).*
4. *Select the minimum value γ_η from the $\hat{p}_N = \hat{p}_N(\gamma)$ function so that $\hat{p}_N(\gamma_\eta) = 1$ is satisfied $\forall \gamma \ge \gamma_\eta$. γ_η is the estimate of the robustness index $\bar{\gamma}$.*

Note that with a simple algorithm we are able to estimate in polynomial time the robustness degree $\bar{\gamma}$ of a generic neural network. The accuracy in estimating $\bar{\gamma}$ can be made arbitrarily good at the expense of a larger number of samples as suggested by Chernoff's bound.

APPLYING THE METHODOLOGY TO STUDY THE ROBUSTNESS OF NEURAL NETWORKS

In the experimental section we show how the robustness index for neural networks can be computed and how it can be used to characterise a neural model. After having presented and experimentally justified the theory supporting Randomised Algorithms, we will focus on the following problems:

- test the robustness of a given static neural network (robustness analysis);
- study the relationships between the robustness of a static neural network and the number of hidden units (structure redundancy);
- analyse the robustness of recurrent neural networks (robustness/stability analysis).

In the following experiments we consider perturbations affecting weights and biases of a neural network defined in D and subject to uniform distributions. Here,

a perturbation Δ_i affecting a generic weight w_i must be intended as a relative perturbation with respect to the weight magnitude according to the multiplicative perturbation model $w_{i,p} = w_i(1+\Delta_i), \forall i = 1, n$. A $t\%$ perturbation implies that Δ_i is drawn from a symmetrical uniform distribution of extremes

$$\left[-\frac{t}{100}, \frac{t}{100}\right];$$

a 5% perturbation affecting weights and biases composing vector $\hat{\theta}$ implies that each weight/bias is affected by an independent perturbation extracted from the [-0.05, 0.05] interval and applied to the nominal value according to the multiplicative perturbation model.

Experiment 1: *The impact of ε, δ and N on the evaluation of the robustness index*

The reference application to be learned is the simple error-free function

$$y = -x \cdot sin(x^2) + \frac{e^{-0.23 \cdot x}}{1+x^4}, \quad x \in [-3,3]$$

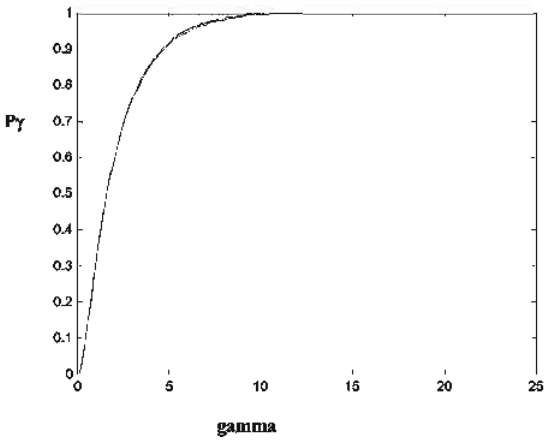
A set of 41 training data have been extracted from the function domain according to a uniform distribution. We considered static feedforward neural networks with hidden units characterised by a hyperbolic tangent activation function and a single linear output. Training was accomplished by considering a Levenberg-Marquardt algorithm applied to an MSE training function; a test set was considered during the training phase to determine the optimal stopping point so as to monitor the upsurgence of overfitting effects.

We discovered that all neural networks with at least 6 hidden units are able to solve the function approximation task with excellent performance.

In this experiment we focus the attention on the neural network characterised by 10 hidden units. After training we run the robustness algorithm by considering 7.5% of perturbations (weights are affected by perturbations up to 7.5% of their magnitude) and we chose $\varepsilon = 0.02$ and $\delta = 0.01$ from which we have to extract N=6624 samples from D. We carried out three extractions of N samples and, for each set, we computed the related $\hat{p}_N = \hat{p}_N(\gamma)$ curve.

The $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Figure 1. As we can see the curves are very close to each other. In fact, we know from the theory, that the estimated probability belongs to a neighbourhood of the true one according to the

Figure 1: $\hat{p}_N = \hat{p}_N(\gamma)$ for three different runs



$\hat{p}_N - 0.02 \le p_\gamma \le \hat{p}_N + 0.02$ relationship. A single curve is therefore enough to characterise the robustness of the envisaged neural network and there is no need to consider multiple runs. By inspecting Figure 1 we obtain that the estimate of the robustness index $\bar{\gamma}$ is γ_η=11 which implies that $U(\Delta) \le 11, \forall \Delta \in D$ with high probability.

We wish now to study the impact of N on $\hat{p}_N = \hat{p}_N(\gamma)$ by considering three runs with different ε and δ according to Table 1.

The $\hat{p}_N = \hat{p}_N(\gamma)$ curves are given in Figure 2. It is interesting to note, at least for the specific application, that even with low values of N, the estimates for $\hat{p}_N = \hat{p}_N(\gamma)$ and γ_η are reasonable and not far from each other. We should anyway extract the number of samples according to Chernoff's inequality.

Experiment 2: *Testing the robustness of a given neural network*

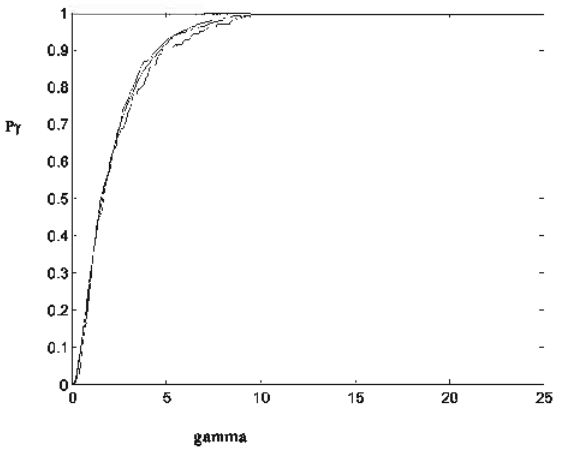
In the second experiment we test the robustness of the 10 hidden units network by considering its behaviour once affected by stronger perturbations (larger D) and, in particular, for perturbations 1%, 3%, 5%, 10%, 30%. We selected ε=0.02 and δ= 0.01.

The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ function corresponding to the different perturbations is given in Figure 3.

Table 1: ε, δ and N

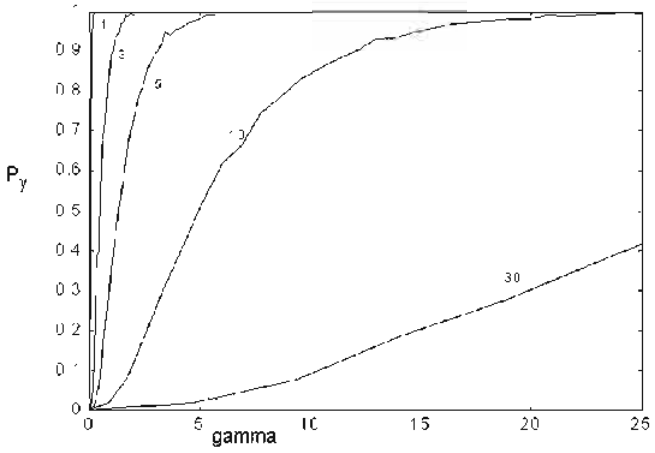
ε	δ	N
0.02	0.01	6624
0.05	0.05	738
0.1	0.1	150

Figure 2: $\hat{p}_N = \hat{p}_N(\gamma)$ for different runs with parameters given in Table 1



Again, from its definition, $\bar{\gamma}$ is the smallest value for which $\hat{p}_\gamma = 1$, $\gamma \geq \bar{\gamma}$; as an example, if we consider the 5% perturbation case, $\bar{\gamma}$ assumes a value around 7. It is obvious, but interesting to point out that, by increasing the strength of perturbation (i.e., by enlarging the extremes of the uniform distribution characterising the *pdf* of D), $\bar{\gamma}$ increases. In fact, stronger perturbations have a worse impact on the performance loss function since the error-affected neural network diverges from the error-free one. Conversely, we see that small perturbations, e.g., the 1% one, induce a very small loss in performance since the robustness index γ_η is very small.

Figure 3: $\hat{p}_\gamma$ as a function of γ for the 10 hidden units neural network



Experiment 3: *Testing the robustness of a hierarchy of performance-equivalent neural networks*

Once we have identified the robustness degree of a neural network solving an application, we can investigate whether it is possible to improve the robustness degree of the application by considering a sort of structural redundancy or not. This issue can be tackled by considering the neural hierarchy M: $M_1 \subseteq M_2 ... \subseteq M_k ...$ where M_k represents a neural network with k hidden units.

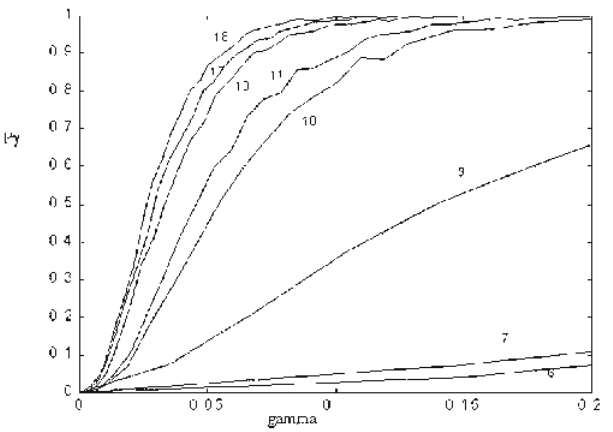
To this end, we consider a set of performance-equivalent neural networks, each of which is able to solve the application with a performance tolerable by the user. All neural networks are characterised by a different topological complexity (number of hidden units).

The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curves parameterised in the number of hidden units are given in Figure 4 in the case of 1% perturbation. We can see that by increasing the number of hidden units, $\bar{\gamma}$ decreases. We immediately realise that neural networks with a reduced number of hidden units are, for this application, less robust than the ones possessing more degrees of freedom. Large networks provide, in a way, a sort of spatial redundancy: information characterising the knowledge space of the neural networks is distributed over more degrees of freedom.

We discovered cases where a larger neural network was less robust than a smaller one: in such a case probably the complex model degenerates into a simpler one.

The evolution of $\bar{\gamma}$ over the number of hidden units parameterised with respect to the different perturbations 5%, 10% and 30% is given in Figure 5. We note that the minimal network, namely the smallest network able to solve the application, is not the more robust one for this application (in fact it possesses large values for the $\bar{\gamma}$ s).

Figure 4: $\hat{p}_\gamma$ *over* γ *and parameterised in the number of hidden units*



This trend—verified also with other applications—suggests that the robustness degree of the neural network improves on the average by increasing the number of hidden units (spatial redundancy). Anyway, with a small increase in the topological complexity (e.g., by considering the 13 hidden units model instead of the 6 one), we obtain a significant improvement according to the robustness level. There is no need to consider more complex neural networks since the improvement in robustness is small.

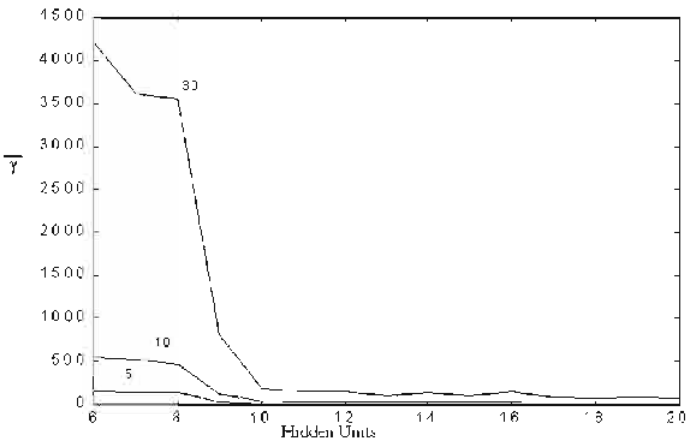
Experiment 4: *Testing the robustness of recurrent neural networks*

The goal of the last experiment is to study the robustness/stability of recurrent neural networks with the suggested theory. The chosen application refers to the identification of the open-loop stable, nonlinear, continuous system suggested in Norgaard (2000). The input and the corresponding output sequence of the system to be identified is given in Figure 6.

We first considered an NOE recurrent neural network with 5 hidden units characterised by the regressor vector $\varphi = [u(t-1), u(t-2), \hat{y}(t-1), \hat{y}(t-2)]$. The non-linear core of the neural network is a static regression type neural network as the one considered in the function approximation experiments. The topology of the NOE network is given in figure 7.

Once trained, the network we applied the methodology to estimates the robustness of the recurrent model. The $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve, evaluated with $\varepsilon = \delta = 0.05$, for the 0.1% perturbation case is given in Figure 8. As we could have expected, differently from the static function approximation application, the recurrent NOE neural network is sensitive even to small perturbations affecting the knowledge space of the network.

Figure 5: $\bar{\gamma}$ *as function of the hidden units,* ε *=0.04,* δ *=0.01*



We identified the dynamic system with a NARMAX neural network characterised by 5 hidden units and the structure given in Figure 9. For such topology we selected the regressor vector

$$\varphi = [u(t-1), u(t-2), y(t-1), y(t-2), e(t-1), e(t-2)]$$

Figure 10 shows the $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve. It is interesting to note that the NARMAX neural network is less robust than the corresponding NOE model. The basic reason for such behaviour is due to the fact that the recurrent model does not receive directly as input the fed-back network output but only the residual *e*.

During training the NOE model must somehow learn more deeply the concept of stability since even small variations of weights associated with the training phase weights update would produce a trajectory diverging from the system output to be mimicked. This effect is due to the pure fed-back structure of the NOE model which

Figure 6: The input and the corresponding output of the dynamic system

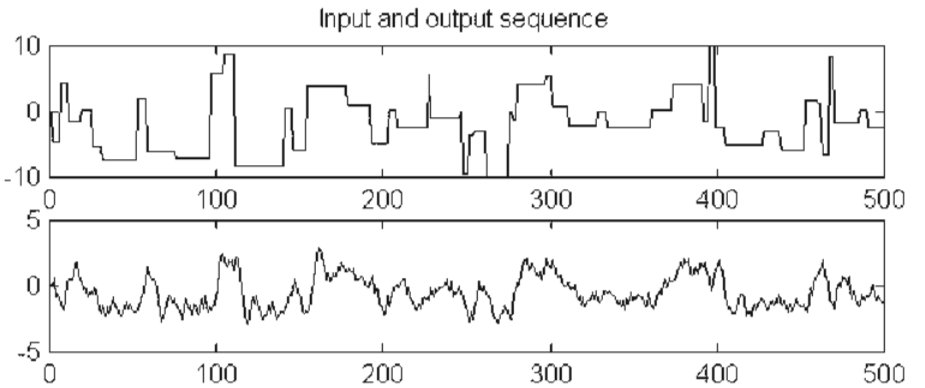


Figure 7: The considered NOE neural network

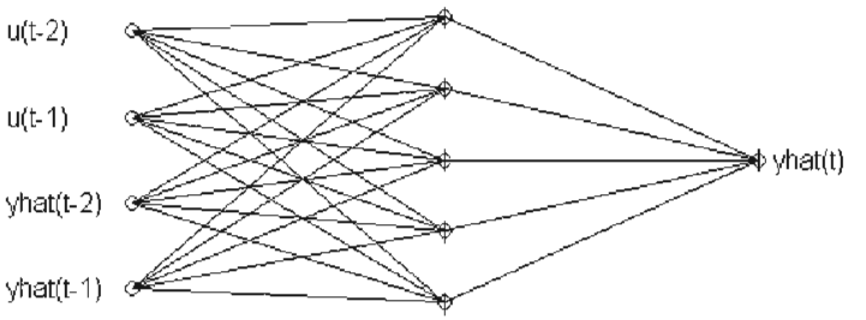


Figure 8: The $\hat{p}_\gamma$ function for the NOE neural network

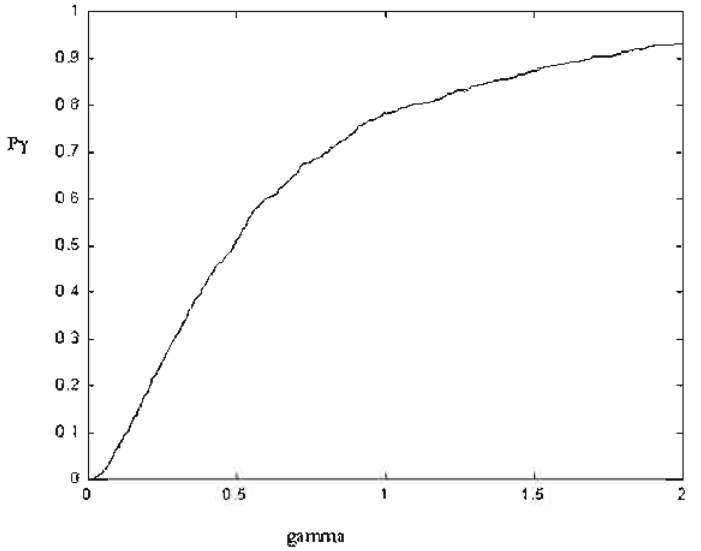
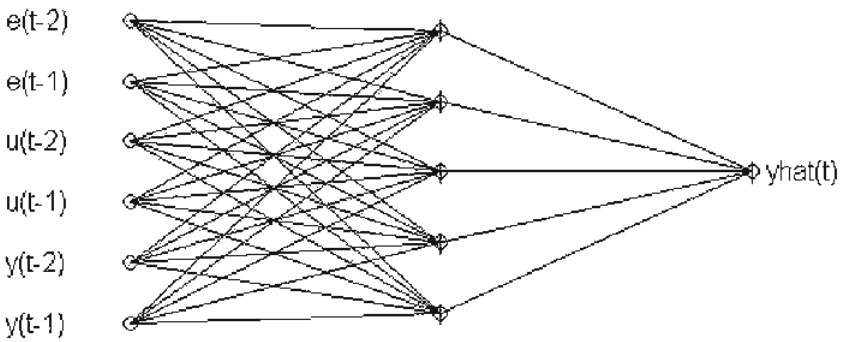
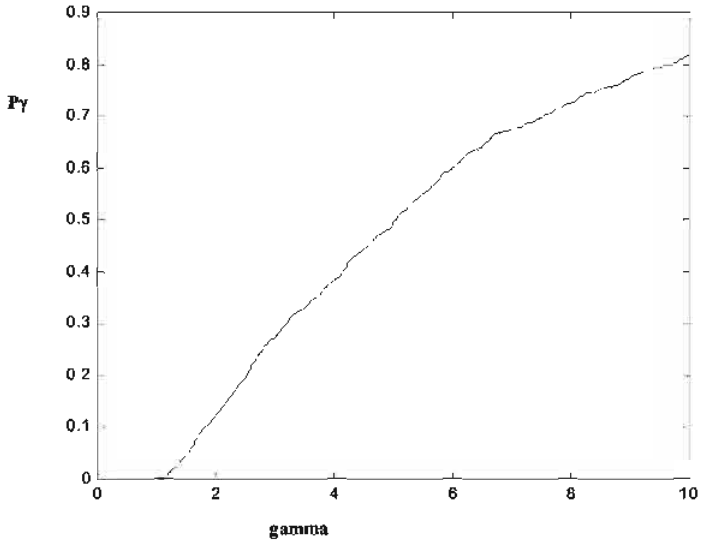


Figure 9: The considered NARMAX neural network



receives as inputs past predicted output and not direct information from the process. Interestingly, this requires the neural model to implicitly learn, during the training phase, the concept of robustness as proven by the $\hat{p}_\gamma = \hat{p}_\gamma(\gamma)$ curve. Conversely, the NARMAX model has a smoother and less complex training phase since it receives fresh information directly from the process (y values) which help the neural model to be stable. As such, the training procedure will not search for weights configuration particularly robust since small deviations, which could make the system unstable, will be directly stabilised by the true information coming from the process.

Figure 10: The $\hat{p}_\gamma$ function for the NARMAX neural network



CONCLUSION

The main results of the chapter can be summarised as follows. Once given a trained neural network:

- the effects of perturbations affecting the network weights can be evaluated regardless of the topology and structure of the neural network, the strength of the perturbation by considering a probabilistic approach;
- the robustness/sensitivity analysis can be carried out with a Poly-time algorithm by resorting to Randomised Algorithms;
- the analysis is independent from the figure of merit considered to evaluate the loss in performance induced by the perturbations.

REFERENCES

Alippi, C. (2002). Randomized Algorithms: A system-level, Poly-time analysis of robust computation. *IEEE Transactions on Computers.* 51(7).

Alippi, C. & Briozzo, L. (1998). Accuracy vs. precision in digital VLSI architectures for signal processing. *IEEE Transactions on Computers,* 47(4).

Alippi, C., Piuri, V. & Sami, M. (1995). Sensitivity to Errors in Artificial Neural Networks: A Behavioural Approach. IEEE Transactions on Circuits and Systems: Part 1, 42(6).

Bai, E., Tempo, R. & Fu, M. (1997). Worst-case properties of the uniform distribution and randomized algorithms for robustness analysis. *IEEE-American Control Conference,* Albuquerque, NM.

Bhattacharyya, S.P, Chapellat, H. & Keel, L.H. (1995). *Robust Control: The Parametric Approach.* Englewoods Cliffs, NJ: Prentice Hall.

Calafiore, G., Dabbene, F. & Tempo, R. (1999). Uniform sample generation of vectors in lp balls for probabilistic robustness analysis. In *Recent Advances in Control,* Springer-Verlag.

Chen, X. & Zhou, K. (1997). On the probabilistic characterisation of model uncertainty and robustness. *Proceedings IEEE-36th Conference on Decision and Control,* San Diego, CA.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. 23.

Djavdan, P., Tulleken, H., Voetter, M., Verbruggen, H. & Olsder, G. (1989). Probabilistic Robust Controller Design. *Proceedings IEEE-28th Conference on Decision and Control,* Tampa, FL.

Dundar, G., Rose, K. (1995). The effects of Quantization on Multilayer Neural Networks, IEEE Transactions of Neural Networks. 6(6).

Hassoun, M.H. (1995). *Fundamentals of Artificial Neural Networks,* The MIT Press.

Hertz, J., Krogh, A. & Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation,* Addison-Wesley Publishing Co.

Holt, J. & Hwang, J. (1993). Finite precision error analysis of neural network hardware implementations. IEEE Transactions on Computers. 42(3).

Ljung, L. (1987). *System Identification, theory for the user,* Prentice-Hall.

Ljung, L. Sjoberg, J. & Hjalmarssoon, H. (1996). On neural networks model structures in system identification. In *Identification, Adaptation, Learning.* NATO ASI series.

Norgaard, M. (2000). Neural Network-Based System Identification Toolbox.

Piché, S. (1995). The selection of weights accuracies for Madalines. IEEE Transactions on Neural Networks. 6(2).

Stevenson, M., Winter, R. & Widrow, B. (1990). Sensitivity of Feedforward neural networks to weights errors, IEEE Transactions on Neural Networks. 1(1).

Tempo, R. & Dabbene, F. (1999). Probabilistic Robustness Analysis and Design of Uncertain Systems. Progress in Systems and Control Theory. 25.

Vidyasagar, M. (1996). *A Theory of Learning and Generalisation with Applications to Neural Networks and Control Systems.* Berlin: Springer-Verlag.

Vidyasagar, M. (1998). Statistical learning Theory and Randomized algorithms for Control. IEEE-Control systems.

Chapter III

Helicopter Motion Control Using a General Regression Neural Network

T.G.B. Amaral
Superior Technical School of Setúbal – IPS, Portugal

M.M. Crisóstomo
University of Coimbra, Portugal

V. Fernão Pires
Superior Technical School of Setúbal – IPS, Portugal

ABSTRACT

This chapter describes the application of a general regression neural network (GRNN) to control the flight of a helicopter. This GRNN is an adaptive network that provides estimates of continuous variables and is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. An important reason for using the GRNN as a controller is the fast learning capability and its non-iterative process. The disadvantage of this neural network is the amount of computation required to produce an estimate, which can become large if many training instances are gathered. To overcome this problem, it is described as a clustering algorithm to produce representative exemplars from a group of training instances that are close to one another reducing the computation amount to obtain an estimate. The reduction of training data used by the GRNN can make it possible to separate the obtained representative

exemplars, for example, in two data sets for the coarse and fine control. Experiments are performed to determine the degradation of the performance of the clustering algorithm with less training data. In the control flight system, data training is also reduced to obtain faster controllers, maintaining the desired performance.

INTRODUCTION

The application of a general regression neural network to control a non-linear system such as the flight of a helicopter at or near hover is described. This general regression neural network in an adaptive network that provides estimates of continuous variables and is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another. The automatic flight control system, through the longitudinal and lateral cyclic, the collective and pedals are used to enable a helicopter to maintain its position fixed in space for a long period of time. In order to reduce the computation amount of the gathered data for training, and to obtain an estimate, a clustering algorithm was implemented. Simulation results are presented and the performance of the controller is analysed.

HELICOPTER MOTION CONTROL

Recently, unmanned helicopters, particularly large-scale ones, have been expected not only for the industrial fields such as agricultural spraying and aerial photography, but also for such fields as observation, rescuing and fire fighting. For monotonous and dangerous tasks, an autonomous flight control of the helicopter is advantageous.

In general, the unmanned helicopter is an example of an intelligent autonomous agent. Autonomous flight control involves some difficulties due to the following:

- it is non-linear;
- flight modes are cross-coupled;
- its dynamics are unstable;
- it is a multivariate (i.e., there are many input-output variables) system;
- it is sensitive to external disturbances and environmental conditions such as wind, temperature, etc;
- it can be used in many different flight modes (e.g., hover or forward flight), each of which requires different control laws;
- it is often used in dangerous environments (e.g., at low altitudes near obstacles).

These characteristics make the conventional control difficult and create a challenge to the design of intelligent control systems.

For example, although helicopters are non-linear systems, NN controllers are capable of controlling them because they are also inherently non-linear. The instabilities that result from time delays between changes in the system input and output can be addressed with the previous learning of the network with a set of data that represents the pilots knowledge to stabilize the helicopter. Linear NN can be implemented to compensate the cross-couplings between control inputs, mainly when the helicopter makes a significant change in its flight.

Therefore, a supervised general regression neural network can be used to control the flight modes of an unmanned helicopter. The regression is the least-mean-squares estimation of the value of a variable based on data samples. The term *general regression* implies that the regression surface is not restricted by being linear. If the values of the variables to be estimated are future values, the general regression network (GRNN) is a predictor. If they are dependent variables related to input variables in a process, system or plant, the GRNN can be used to model the process, system or plant. Once the system is modelled, a control surface can be defined in terms of samples of control variables that, given a state vector of the system, improve the output of the system. If a GRNN is trained using these samples, it can estimate the entire control surface, becoming a controller. A GRNN can be used to map from one set of sample points to another. If the target space has the same dimension as the input space, and if the mapping is one-to-one, an inverse mapping can easily be formed using the same examples. When the variables to be estimated are for intermediate values between given points, then the GRNN can be used as an interpolator.

In all cases, the GRNN instantly adapts to new data points. This could be a particular advantage for training robots to emulate a teacher or for any system whose model changes frequently.

SYSTEM MODELLING

The helicopter control is one of the popular non-linear educational control problems. Due to its highly non-linear dynamics, it gives the possibility to demonstrate basic features and limits of non-linear control concepts. Sugeno (1997, 1998) developed a fuzzy-logic based control system to replace the aircraft's normal set of control inputs. Other researchers, such has Phillips et al. (1994), Wade et al (1994), and Wade and Walker (1994), have developed fuzzy logic flight controls describing systems that include mechanisms for discovering and tuning fuzzy rules in adaptive controllers. (Larkin, 1984) described a model of an autopilot controller based on fuzzy algorithms. An alternative approach to real-time control of an

autonomous flying vehicle based on behavioral, or reactive, approach is proposed by Fagg et al. (1993). A recurrent neural network used to forward modeling of helicopter flight dynamics was described by Walker and Mo (1994). The NN-based controllers can indirectly model human cognitive performance by emulating the biological processes underlying human skill acquisition.

The main difference between NN-based controllers and conventional control systems is that, in the NN case, systems are built from indirectly representations of control knowledge similar to those employed by skilled humans, while in the conventional design case, a deep analytical understanding of the system dynamics is needed. The ability of humans to pilot manned helicopters with only the qualitative knowledge indicate that NN-based controllers with similar capabilities can also be developed.

The helicopter can be modelled as a linear system around trim points, i.e., a flight with no accelerations and no moments. The state space equations are a natural form, which can represent the helicopter motion. The general mathematical model is given by:

$$\dot{x} = Ax + Bu_c$$

$$y = Cx + Du_c$$

where x, u_c *and* y are the state vector, control vector and output vector, respectively.

The helicopter used to simulate the flight in hover position was a single main rotor helicopter of 15,000 pounds. The control and state vectors are defined as:

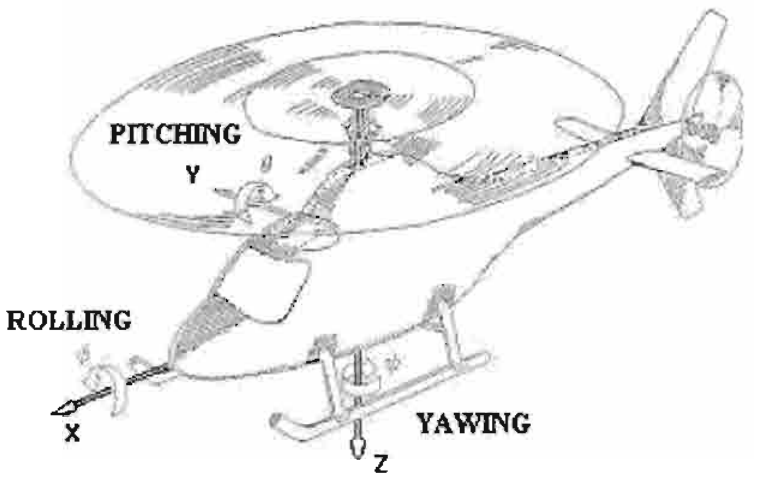
$$u_c^T = [\delta_a \ \delta_b \ \delta_c \ \delta_d] \tag{1}$$

$$x^T = [u \ v \ w \ p \ q \ r \ \phi \ \theta \ \varphi \ \mathrm{x} \ y \ z] \tag{2}$$

where

δ_a is the collective control [*inches*];
δ_b and δ_c are the longitudinal and lateral cyclic controls, respectively [*inches*];

Figure 1: Helicopter coordinates



δ_d is the pedal control [*inches*];
u, v and w are the perturbation linear velocities [*ft*/sec];
p, q and r are the perturbation angular velocities [*rad*/sec];
ϕ, θ and φ are the perturbation euler angles for roll, pitch and yaw [*rad*];
x, y and z are the perturbation linear displacements over the ground [*ft*].

Figure 1 shows the coordinate system to describe the motion of the helicopter. The origin of the helicopter axes is placed on the center of gravity.

The thrust of the main rotor, thus mainly the vertical acceleration, is controlled by the collective control (δ_a). The pitching moment, that is, nose pointing up or down, is controlled by the longitudinal cyclic control (δ_b). The rolling moment, that is, right wing tip down, left wing tip up, and vice versa, is controlled by the lateral cyclic control (δ_c). The yawing moment, that is, nose left and right, is controlled by the pedal control (δ_d).

The corresponding differential equations that represent the behavior of the helicopter in hover position are:

$$\frac{du}{dt} = -0.069u - 0.032v + 116.8p + 1168.5q - 6.15r - 32.19\theta + 0.118\delta_a$$
$$- 2.79\delta_b - 0.25\delta_c + 0.0043\delta_d$$

$$\frac{dv}{dt} = 0.017u - 0.085v - 0.0021w - 430.5p + 381.3q + 30.75r + 32.14\phi + 0.023\theta - 0.14\delta_a - \delta_b + 0.665\delta_c - 1.39\delta_d$$

$$\frac{dw}{dt} = -0.0021v - 0.257w + 7.99p + 46.74q + 135.3r + 1.85\phi - 0.404\theta - 9.23\delta_a - 0.107\delta_b - 0.01\delta_c$$

$$\frac{dp}{dt} = 0.45u - 0.687v - 0.0021w - 6027.2p + 5043.16q + 664.2r - 1.82\delta_a - 13.7\delta_b + 8.58\delta_c - 5.15\delta_d$$

$$\frac{dq}{dt} = 0.665u + 0.429v - 0.043w - 1537.5p - 15744.5q - 12.3r - 0.966\delta_a + 37.13\delta_b + 3.43\delta_c + 0.75\delta_d$$

$$\frac{dr}{dt} = -0.0214u + 0.515v + 0.0064w - 369.0p - 44.28q - 1266.9r + 25.97\delta_a - 0.15\delta_b + 0.075\delta_c + 40.78\delta_d$$

$$\frac{d\phi}{dt} = p$$

$$\frac{d\theta}{dt} = q$$

$$\frac{d\varphi}{dt} = r$$

$$\frac{dx}{dt} = u$$

$$\frac{dy}{dt} = v$$

$$\frac{dz}{dt} = w$$

Since each motion is not independent of δ_a, δ_b, δ_c and δ_d, there exists a cross-coupling.

Figure 2 shows the root locus for the model described above. Figure 2(a) shows the root locus, considering the collective control as the input and the vertical displacement as the output. In Figure 2(c), the longitudinal cyclic and the forward displacement are the input and the output, respectively. Figure 2(e) shows the root locus considering the lateral cyclic as the input and the lateral displacement as the output. Figures 2(b), (d) and (f) show the zoom of the region near the imaginary axis as well as the roots that dominate the transient response. In general, the contribution

Figure 2: Root locus of the helicopter model

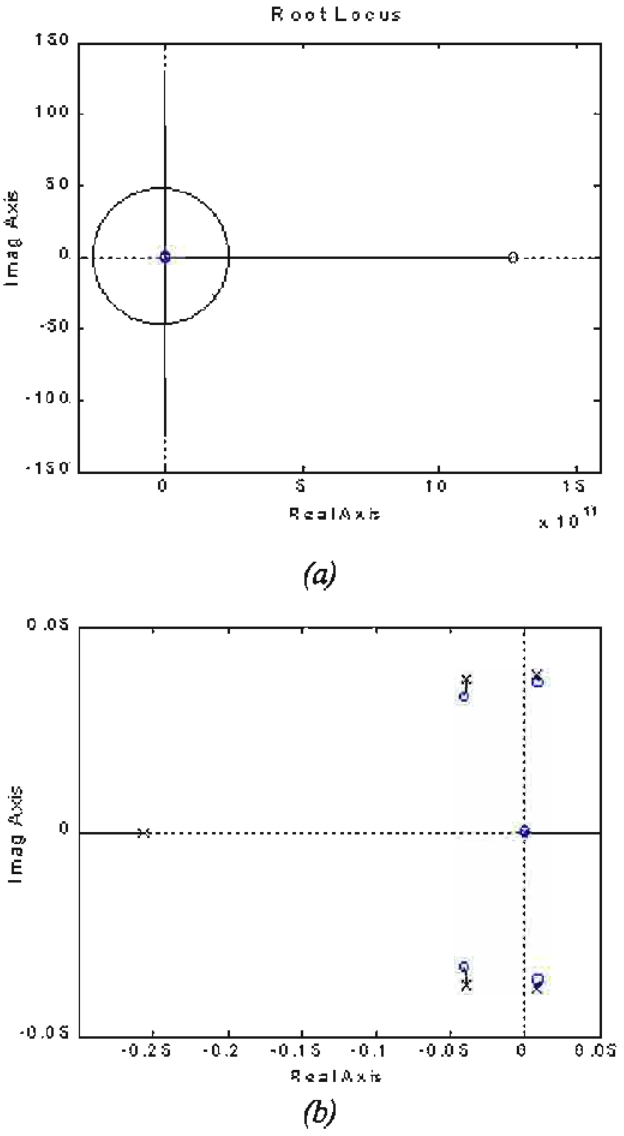
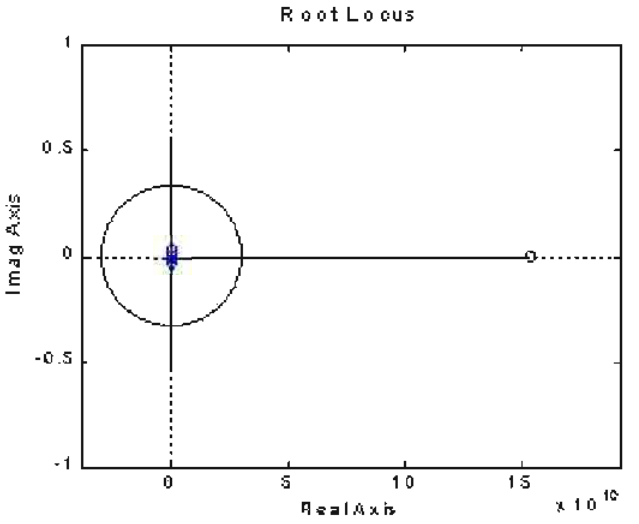
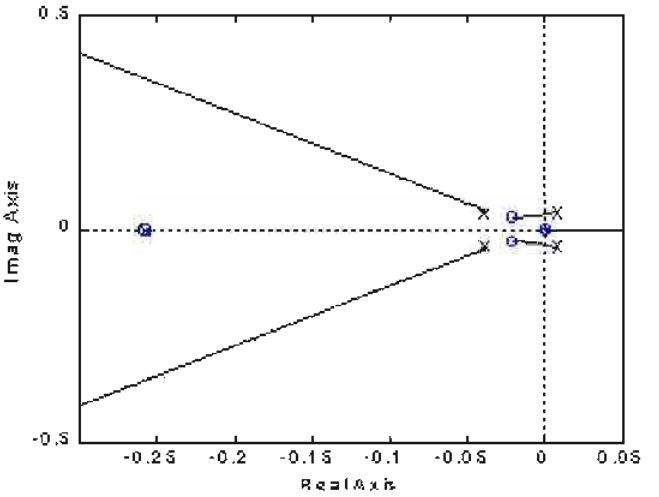


Figure 2: Root locus of the helicopter model (continued)



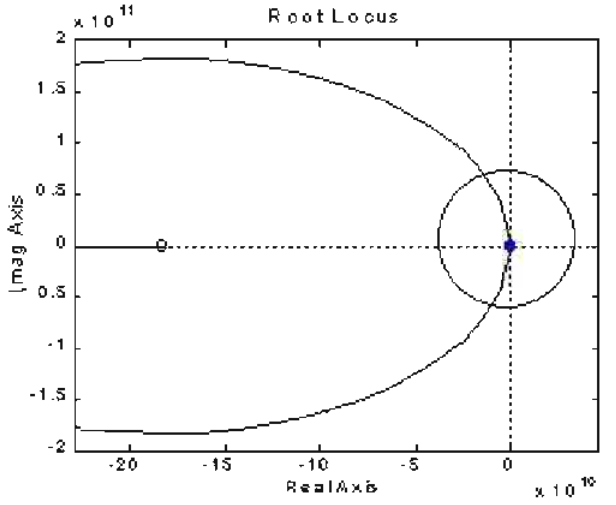
(c)



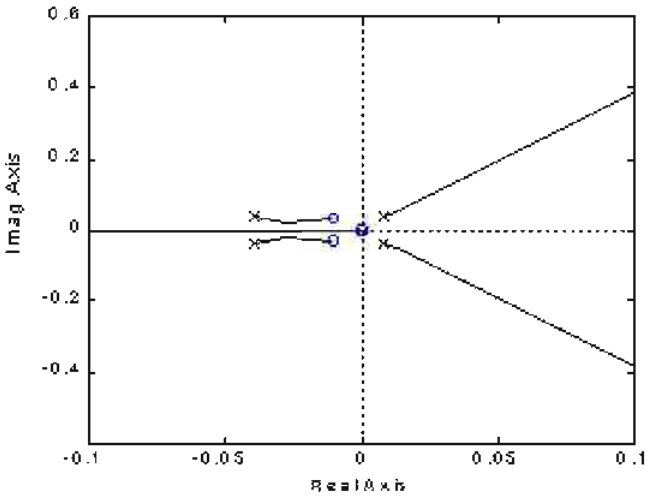
(d)

in the time response of roots that lie relatively far to the left in the s-plane will be small. These three Figures clearly show that some of the eigenvalues corresponding to the helicopter model are in the right side of the s-plane, with positive real-part values, making the system unstable.

Figure 2: Root locus of the helicopter model (continued)



(e)



(f)

GENERAL REGRESSION NEURAL NETWORK

The generalized regression neural networks are memory-based feed-forward networks originally developed in the statistics literature by Nadaraya (1964) and known as Nadaraya-Watson kernel regression. Then the GRNN was 're-discovered' by Specht (1991) and Chen, C. (1996), with the desired capability of