

Artificial Neural Networks in Finance and Manufacturing



Joarder Kamruzzaman, Rezaul Begg and Ruhul Sarker

Artificial Neural Networks in Finance and Manufacturing

Joarder Kamruzzaman
Monash University, Australia

Rezaul K. Begg
Victoria University, Australia

Ruhul A. Sarker
University of New South Wales, Australia



IDEA GROUP PUBLISHING

Hershey • London • Melbourne • Singapore

Acquisitions Editor: Michelle Potter
Development Editor: Kristin Roth
Senior Managing Editor: Amanda Appicello
Managing Editor: Jennifer Neidig
Copy Editor: Chuck Pizar
Typesetter: Cindy Consonery
Cover Design: Lisa Tosheff
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
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 0609
Web site: <http://www.eurospanonline.com>

Copyright © 2006 by Idea Group Inc. All rights reserved. No part of this book may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.

Product or company names used in this book are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Neural networks in finance and manufacturing / Joarder Kamruzzaman, Rezaul Begg and Ruhul Sarker, editors.

p. cm.

Summary: "This book presents a variety of practical applications of neural networks in two important domains of economic activity: finance and manufacturing"--Provided by publisher. Includes bibliographical references and index.

ISBN 1-59140-670-6 (hardcover) -- ISBN 1-59140-671-4 (softcover) -- ISBN 1-59140-672-2 (ebook)

1. Neural networks (Computer science)--Economic aspects. 2. Neural networks (Computer science)--Industrial applications. 3. Finance--Computer simulation. 4. Manufacturing processes--Computer simulation. I. Kamruzzaman, Joarder. II. Begg, Rezaul. III. Sarker, Ruhul A.

HD30.2.N48 2006

332.0285'632--dc22

2006003560

British Cataloguing in Publication Data

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

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

Artificial Neural Networks in Finance and Manufacturing

Table of Contents

Preface vi

SECTION I: INTRODUCTION

Chapter I.

Artificial Neural Networks: Applications in Finance and Manufacturing 1

Joarder Kamruzzaman, Monash University, Australia

Ruhul A. Sarker, University of New South Wales, Australia

Rezaul K. Begg, Victoria University, Australia

Chapter II.

Simultaneous Evolution of Network Architectures and Connection Weights in Artificial Neural Networks 28

Ruhul A. Sarker, University of New South Wales, Australia

Hussein A. Abbass, University of New South Wales, Australia

SECTION II: ANNs IN FINANCE

Chapter III.

Neural Network-Based Stock Market Return Forecasting Using Data Mining for Variable Reduction 43

David Enke, University of Missouri — Rolla, USA

Chapter IV.

Hybrid-Learning Methods for Stock Index Modeling 64

Yuehui Chen, Jinan University, P.R. China

Ajith Abraham, Chung-Ang University, Republic of Korea

Chapter V.
Application of Higher-Order Neural Networks to Financial Time-Series Prediction 80
John Fulcher, University of Wollongong, Australia
Ming Zhang, Christopher Newport University, USA
Shuxiang Xu, University of Tasmania, Australia

Chapter VI.
Hierarchical Neural Networks for Modelling Adaptive Financial Systems 109
Masoud Mohammadian, University of Canberra, Australia
Mark Kingham, University of Canberra, Australia

Chapter VII.
Forecasting the Term Structure of Interest Rates Using Neural Networks 124
Sumit Kumar Bose, Indian Institute of Management, India
Janardhanan Sethuraman, Indian Institute of Management, India
Sadhalexmi Raipet, Indian Institute of Management, India

Chapter VIII.
Modeling and Prediction of Foreign Currency Exchange Markets 139
Joarder Kamruzzaman, Monash University, Australia
Ruhul A. Sarker, University of New South Wales, Australia
Rezaul K. Begg, Victoria University, Australia

Chapter IX.
Improving Returns on Stock Investment through Neural Network Selection 152
Tong-Seng Quah, Nanyang Technological University, Republic of Singapore

SECTION III: ANNs IN MANUFACTURING

Chapter X.
Neural Networks in Manufacturing Operations 165
Eldon Gunn, Dalhousie University, Canada
Corinne MacDonald, Dalhousie University, Canada

Chapter XI.
High-Pressure Die-Casting Process Modeling Using Neural Networks 182
M. Imad Khan, Deakin University, Australia
Saeid Nahavandi, Deakin University, Australia
Yakov Frayman, Deakin University, Australia

Chapter XII.
Neural Network Models for the Estimation of Product Costs: An Application in the Automotive Industry 199
Sergio Cavalieri, Università degli Studi di Bergamo, Italy
Paolo Maccarrone, Politecnico di Milano, Italy
Roberto Pinto, Università degli Studi di Bergamo, Italy

Chapter XIII.	
A Neural-Network-Assisted Optimization Framework and Its Use for Optimum-Parameter Identification	221
<i>Tapabrata Ray, University of New South Wales, Australia</i>	
Chapter XIV.	
Artificial Neural Networks in Manufacturing: Scheduling	236
<i>George A. Rovithakis, Aristotle University of Thessaloniki, Greece</i>	
<i>Stelios E. Perrakis, Technical University of Crete, Greece</i>	
<i>Manolis A. Christodoulou, Technical University of Crete, Greece</i>	
Chapter XV.	
Recognition of Lubrication Defects in Cold Forging Process with a Neural Network	262
<i>Bernard F. Rolfe, Deakin University, Australia</i>	
<i>Yakov Frayman, Deakin University, Australia</i>	
<i>Georgina L. Kelly, Deakin University, Australia</i>	
<i>Saeid Nahavandi, Deakin University, Australia</i>	
About the Authors	276
Index	284

Preface

Artificial neural networks (ANNs) have attracted increasing attentions in recent years for solving many real-world problems. ANNs have been successfully applied in solving many complex problems where traditional problem-solving methods have failed or proved insufficient. With significant advances in processing power, neural networks research has been able to address problems that were often tackled by using simplified assumptions in the past. This has resulted in a wealth of new approaches based on neural networks in many areas, particularly in finance and manufacturing. This is evidenced by the exponential growth of scientific literature covering applications of neural networks in these areas.

Research and development works in ANNs are still growing rapidly due to an increasing number of successful applications of these techniques in diverse disciplines. This book is intended to cover basic theory and concepts of neural networks followed by recent applications of such techniques in finance and manufacturing. The book contains 15 chapters divided into three parts as follows:

- Section I: Introduction
- Section II: ANNs in Finance
- Section III: ANNs in Manufacturing

Section I gives an introduction to neural networks and their basic components. The individual neuron operation, network architecture, and training algorithms are discussed in the first part of Chapter I. The second part of this chapter provides a brief review of ANN applications in finance and manufacturing. Chapter II introduces one of the latest research areas in this field, which is evolving ANNs. In this chapter, the authors investigate the simultaneous evolution of network architectures and connection weights in ANNs. In simultaneous evolution, they use the well-known concept of multiobjective optimization and subsequently evolutionary multiobjective algorithms to evolve ANNs.

The results are promising when compared with the traditional ANN algorithms. It is expected that this methodology would provide better solutions to many applications of ANNs.

Section II of this book consists of seven chapters on ANN applications in the financial domain. Chapter III investigates the use of ANNs for stock market return forecasting. The authors examined neural network models, for level estimation and classification, to provide an effective forecasting of future values. A cross-validation technique was also employed to improve the generalization ability of the models. The results show that the classification models generate higher accuracy in forecasting ability than the buy-and-hold strategy, as well as those guided by the level-estimation-based forecasts of the neural network and benchmark linear regression models.

In Chapter IV, the authors investigate the development of novel reliable and efficient techniques to model the seemingly chaotic behavior of stock markets. They considered the flexible neural tree algorithm, a wavelet neural network, local linear wavelet neural network, and finally a feed-forward artificial neural network. The particle swarm optimization algorithm optimized the parameters of the different techniques. This chapter briefly explains how the different learning paradigms can be formulated using various methods and then investigated as to whether they can provide the required level of performance. Experimental results revealed that all the models considered could represent the stock indices behavior very accurately.

In many situations, financial time-series data is characterized by nonlinearities, discontinuities, and high-frequency multi-polynomial components. The conventional ANNs have difficulty in modeling such complex data. Chapter V provides an appropriate approach that is capable of extracting higher-order polynomial coefficients in the data. The authors later incorporated piecewise continuous activation functions and thresholds, and as a result, they are capable of modeling discontinuous (or piecewise continuous) data with a higher degree of accuracy. The performance of their approach was tested using representative financial time-series data such as credit ratings and exchange rates.

In Chapter VI, an intelligent Hierarchical Neural Network system for prediction and modeling of interest rates is presented. The proposed system was developed to model and predict 3-month (quarterly) interest-rate fluctuations. The system was further trained for 6-month and 1-year periods. The authors nicely analyzed the accuracy of prediction produced by their approach.

Although many works exist on the issue of modeling the yield curve, there is virtually no mention in the literature on the issue of forecasting the yield curve. In Chapter VII, the authors applied neural networks for the purpose of forecasting the zero-coupon yield curve. First, the yield curve was modeled from the past data using the famous Nelson-Siegel model. Then, forecasting of the various parameters of the Nelson-Siegel yield curve was performed using two different techniques — the multilayer perceptron and generalized feed-forward network. The forecasted Nelson-Siegel parameters were then used to predict the yield and the price of the various bonds. Results show the superiority of generalized feed-forward network over the multilayer perceptron for the purposes of forecasting the term structure of interest rates.

In Chapter VIII, the authors investigated an ANN-based prediction modeling of foreign currency rates using three different learning algorithms. The models were trained from

historical data using five technical indicators to predict six currency rates against the Australian dollar. The forecasting performance of the models was evaluated using a number of widely used statistical metrics. Results show that significantly better prediction can be made using simple technical indicators without extensive knowledge of the market data. The trading profitability of the neural-network-based forex model over a forecasted period was also analyzed.

Chapter IX deals with another important financial application — analysis of stock return for investment. The author applies neural networks for stock selection in the Singapore market. This chapter shows that neural networks are able to infer the characteristics of performing stocks from the historical patterns. The performance of stocks is reflective of the profitability and quality of management of the underlying company. Such information is reflected in financial and technical variables. A neural network based on a moving window selection system is used to uncover the intricate relationships between the performance of stocks and the related financial and technical variables. Historical data such as financial variables (inputs) and performance of the stock (output) is used to train the model. Experimental results show the model is capable of selecting stocks that yield better investment return.

Section III of the book contains six chapters on ANN applications in a manufacturing environment. The first chapter in this part (Chapter X) is a review chapter that discusses a number of examples of the use of neural networks in manufacturing operations.

Chapter XI presents an application of neural networks to the industrial-process modeling of high-pressure die casting. The model was implemented in two stages. The first stage was to obtain an accurate model of the die-casting process using a feed-forward multilayer perceptron from the process-condition monitoring data. The second stage was to evaluate the effect of different process parameters on the level of porosity in castings by performing sensitivity analysis. The results obtained were very encouraging to model die-casting process accurately.

The estimation of the unit production cost of a product during its design phase can be extremely difficult, especially if information on similar products previously produced is missing. In Chapter XII, the authors applied ANNs to determine the correlation between a product's cost and its characteristics. The test results seemed very good.

In Chapter XIII, a framework for design optimization is introduced that makes use of neural-network-based surrogates in lieu of actual analysis to arrive at optimum process parameters. The performance of the algorithm was studied using a number of mathematical benchmarks to instill confidence on its performance before reporting the results of a spring-back minimization problem. The results clearly indicate that the framework is able to report optimum designs with a substantially low computational cost while maintaining an acceptable level of accuracy.

In Chapter XIV, a neuro-adaptive scheduling methodology for machines is presented and evaluated by comparing its performance with conventional schedulers. The authors employed a dynamic neural network model and subsequently derived a continuous-time neural network controller and the control-input discretization process that yield the actual dispatching times. The developed algorithm guarantees system stability and controller-signal boundedness and robustness. The algorithm was evaluated on an industrial test case that constitutes a highly nonacyclic deterministic job shop

with extremely heterogeneous part-processing times. The simulation study, employing the idealistic deterministic job-shop abstraction, provided extensive comparison with conventional schedulers over a broad range of raw-material arrival rates and, through the extraction of several performance indices, verified its superb performance in terms of manufacturing system stability and low makespan, low average lead times, work-in-process inventory, and backlogging costs. Eventually, these extensive experiments highlighted the practical value and the potential of the mathematical properties of the proposed neuro-adaptive controller algorithm and its suitability for the control of non-trivial manufacturing cells.

The final chapter (Chapter XV) describes the application of neural networks to recognition of lubrication defects typical to industrial cold forging process. The neural-network-based model learned from different features related to the system was able to recognize all types of lubrication errors to a high accuracy. The overall accuracy of the neural network model was reported to be around 95% with almost uniform distribution of errors between all lubrication errors and the normal condition.

It is hoped that this book will trigger great interest in neural network applications in finance and manufacturing areas, leading to many more articles and books.

Joarder Kamruzzaman, Rezaul Begg, and Ruhul Sarker
Editors

Acknowledgments

We would like to express our gratitude to the contributors without whose submissions this book would not have been published. All of the chapters in this book have undergone a *peer-review* process with each chapter being independently refereed by at least two reviewers in addition to an editorial review by one of the editors. We owe a great deal to these reviewers who reviewed one or more chapters and gave the authors and editors the much needed guidance. Also, we would like to thank those reviewers who could not contribute through authoring chapters to the current book but helped in reviewing chapters within a short period of time.

A special note of thanks must go to all of the staff at Idea Group Inc., whose contributions throughout the whole process from the proposal submission to the final publication have been invaluable. In fact, this book would not have been possible without the ongoing professional support from Senior Academic Technology Editor Dr. Mehdi Khosrow-Pour, Managing Director Ms. Jan Travers, Acquisitions Editor Ms. Renée Davies, Development Editor Ms. Kristin Roth, and Marketing Manager Ms. Amanda Phillips at Idea Group Inc.

We would like to thank our university authorities (Monash University, Victoria University, and the University of New South Wales at the Australian Defence Force Academy) for providing logistic support throughout this project.

Finally, we like to thank our families for their love, support, and patience throughout this project.

Joarder Kamruzzaman, Rezaul Begg, and Ruhul Sarker
Editors

SECTION I:
INTRODUCTION

Chapter I

Artificial Neural Networks: Applications in Finance and Manufacturing

Joarder Kamruzzaman, Monash University, Australia

Ruhul A. Sarker, University of New South Wales, Australia

Rezaul Begg, Victoria University, Australia

Abstract

The primary aim of this chapter is to present an overview of the artificial neural network basics and operation, architectures, and the major algorithms used for training the neural network models. As can be seen in subsequent chapters, neural networks have made many useful contributions to solve theoretical and practical problems in finance and manufacturing areas. The secondary aim here is therefore to provide a brief review of artificial neural network applications in finance and manufacturing areas.

Introduction

Since the seminal work by Rumelhart, McClelland, and the PDP research group (1986), artificial neural networks (ANNs) have drawn tremendous interest due to the demonstrated successful applications in pattern recognition (Fukumi, Omatu, & Nishikawa 1997), image processing (Duranton, 1996), document analysis (Marinai, Gori, & Soda, 2005), engineering tasks (Jin, Cheu, & Srinivasan, 2002; Zhenyuan, Yilu, & Griffin, 2000), financial modeling (Abu-Mostafa, 2001), manufacturing (Kong & Nahavandi, 2002), biomedical (Nazeran & Behbehani, 2000), optimization (Cho, Shin, & Yoo, 2005), and so on. In recent years, there has been a wide acceptance of ANNs as a tool for solving many financial and manufacturing problems. In finance, domain notable applications are in (1) trading and forecasting including derivative-securities pricing and hedging (Steiner & Wittkemper, 1997), (2) future price estimation (Torsun, 1996), (3) stock performance and selection (Kim & Chun, 1998), (4) foreign exchange rate forecasting (Kamruzzaman & Sarker, 2003), (5) corporate bankruptcy prediction (Atiya, 2001), (6) fraud detection (Smith & Gupta, 2000), and so on. Many commercial software based on ANNs are also available today offering solutions to a wide range of financial problems. Applications in manufacturing includes (1) condition monitoring in different manufacturing operations such as metal forming (Kong & Nahavandi, 2002), drilling (Brophy, Kelly, & Bryne, 2002), turning (Choudhury, Jain, & Rama Rao, 1999), and tool wearing and breaking (Choudhury, Jain, & Rama Rao, 1999; Huang & Chen, 2000), (2) cost estimation (Cavaliere, Maccarrone, & Pinto, 2004), (3) fault diagnosis (Javadpour & Knapp, 2003), (4) parameter selection (Wong & Hamouda, 2003), (5) production scheduling (Yang & Wang, 2000), (6) manufacturing cell formation (Christodoulou & Gaganis, 1998), and (7) quality control (Bahlmann, Heidemann, & Ritter, 1999).

Although developed as a model for mimicking human intelligence into machine, neural networks have excellent capability of learning the relationship between input-output mapping from a given dataset without any knowledge or assumptions about the statistical distribution of data. This capability of learning from data without any *a priori* knowledge makes neural networks particularly suitable for classification and regression tasks in practical situations. In most financial and manufacturing applications, classification and regression constitute integral parts. Neural networks are also inherently nonlinear which makes them more practical and accurate in modeling complex data patterns as opposed to many traditional methods which are linear. In numerous real-world problems including those in the fields of finance and manufacturing, ANN applications have been reported to outperform statistical classifiers or multiple-regression techniques in classification and data analysis tasks. Because of their ability to generalize well on unseen data, they are also suitable to deal with outlying, missing, and/or noisy data. Neural networks have also been paired with other techniques to harness the strengths and advantages of both techniques.

Since the intention of this book is to demonstrate innovative and successful applications of neural networks in finance and manufacturing, this introductory chapter presents a broad overview of neural networks, various architectures and learning algorithms, and some convincing applications in finance and manufacturing and discussion on current research issues in these areas.

Artificial Neural Networks

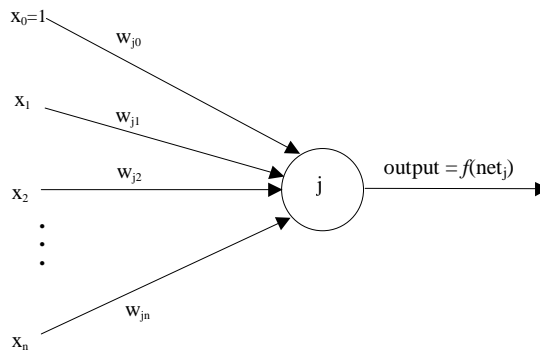
ANNs offer a computational approach that is quite different from conventional digital computation. Digital computers operate sequentially and can do arithmetic computation extremely fast. Biological neurons in the human brain are extremely slow devices and are capable of performing a tremendous amount of computation tasks necessary to do everyday complex tasks, commonsense reasoning, and dealing with fuzzy situations. The underlining reason is that, unlike a conventional computer, the brain contains a huge number of neurons, information processing elements of the biological nervous system, acting in parallel. ANNs are thus a parallel, distributed information processing structure consisting of processing elements interconnected via unidirectional signal channels called connection weights. Although modeled after biological neurons, ANNs are much simplified and bear only superficial resemblance. Some of the major attributes of ANNs are: (a) they can learn from examples and generalize well on unseen data, and (b) are able to deal with situation where the input data are erroneous, incomplete, or fuzzy.

Individual Neuron

The individual processing unit in ANNs receives input from other sources or output signals of other units and produces an output as shown in Figure 1. The input signals (x_i) are multiplied with weights (w_{ji}) of connection strength between the sending unit “ i ” and receiving unit “ j ”. The sum of the weighted inputs is passed through an activation function. The output may be used as an input to the neighboring units or units at the next layer. Assuming the input signal by a vector \mathbf{x} (x_1, x_2, \dots, x_n) and the corresponding weights to unit “ j ” by \mathbf{w}_j ($w_{j1}, w_{j2}, \dots, w_{jn}$), the net input to the unit “ j ” is given by Equation 1. The weight w_{j0} (= b) is a special weight called bias whose input signal is always +1.

$$net_j = \sum_n w_{jn} x_n + w_{j0} = \mathbf{w}_j \mathbf{x} + b \quad (1)$$

Figure 1. An individual unit in a neural network



In general, a neural network is characterized by the following three major components:

- The computational characteristics of each unit, for example, activation function;
- The network architecture; and
- The learning algorithm to train the network.

Activation Function

The computed weighted sum of inputs is transformed into an output value by applying an activation function. In most cases, the activation function maps the net input between -1 to +1 or 0 to 1. This type of activation function is particularly useful in classification tasks. In cases where a neural network is required to produce any real value, linear activation function may be used at the final layer. A network with multiple layers using linear activation function at intermediate layers effectively reduces to a single-layer network. This type of network is incapable of solving nonlinearly separable problems and has limited capability. Since the most real-world problems are nonlinearly separable, nonlinearity in the intermediate layer is essential for modeling complex problems. There are many different activation functions proposed in the literature that are often chosen to be monotonically increasing functions. The followings are the most commonly used activation functions (see Table 1).

Network Architecture

Having defined an individual neuron, the next step is to connect them together. A neural network architecture represents a configuration indicating how the units are grouped together as well as the interconnection between them. There are many different architectures reported in the literature, however, most of these can be divided into two main broad categories: feed-forward and feedback. These architectures are shown in Figure 2. In feed-forward architecture, the information signal always propagates towards the forward direction while in feedback architecture the final outputs are again fed back at the input

Table 1. Commonly used activation functions


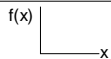
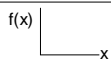
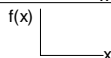
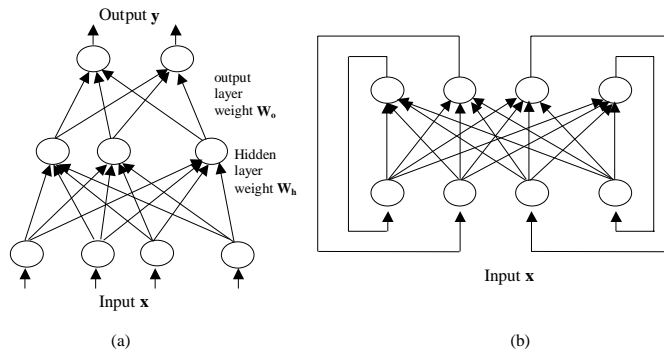
Activation Functions	Mathematical Expression	Graphical Expression
Linear	$f(x) = x$	
Logistic sigmoid	$f(x) = \frac{1}{1 + \exp(-x)}$	
Hyperbolic tangent	$f(x) = \tanh(x)$	
Gaussian	$f(x) = \exp(-x^2/2\sigma^2)$	

Figure 2. (a) Feedforward architecture (b) Feedback architecture



layer. The first layer is known as *input layer*, the last as *output layer*, and any intermediate layer(s) as *hidden layer(s)*. A multiple feedforward layer can have one or more layers of hidden units. The number of units at the input layer and output layer is determined by the problem at hand. Input layer units correspond to the number of independent variables while output layer units correspond to the dependent variables or the predicted values.

While the numbers of input and output units are determined by the task at hand, the numbers of hidden layers and the units in each layer may vary. There are no widely accepted rules for designing the configuration of a neural network. A network with fewer than the required number of hidden units will be unable to learn the input-output mapping, whereas too many hidden units will generalize poorly of any unseen data. Several researchers attempted to determine the appropriate size of hidden units. Kung and Hwang (1988) suggested that the number of hidden units should be equal to the number of distinct training patterns while Arai (1989) concluded that N input patterns required $N-1$ hidden units in a single layer. However, as remarked by Lee (1997), it is rather difficult to determine the optimum network size in advance. Other studies suggested that ANNs generalize better when succeeding layers are smaller than the preceding ones (Kruschke, 1989; Looney, 1996). Although a two-layer network is commonly used in most problem solving approaches, the determination of an appropriate network configuration usually requires many trial and error methods. Another way to select network size is to use constructive approaches. In constructive approaches, the network starts with a minimal size and grows gradually during the training procedure (Fahlman & Lebiere, 1990; Lehtokangas, 2000).

Learning Algorithms

A neural network starts with a set of initial weights and then gradually modifies the weights during the training cycle to settle down to a set of weights capable of realizing the input-output mapping with either no error or a minimum error set by the user. Learning

in neural networks can be *supervised* or *unsupervised*. Supervised learning includes Backpropagation and its variants, Radial Basis Function Neural Network (RBFNN), Probabilistic Neural Network (PNN), Generalized Regression Neural Network (GRNN), and so on. In supervised learning, an input datum is associated with a known output, and training is done in pairs. Unsupervised learning, for example, Self Organizing Map (SOM), Adaptive Resonance Theory (ART), and so on, is used when training sets with known outputs are not available. In the following, we describe some of the widely used ANN learning algorithms.

Backpropagation Algorithm

A recent study (Wong, Lai, & Lam, 2000) has shown that approximately 95% of the reported neural network business applications utilize multilayer feed-forward neural networks with Backpropagation learning algorithm. Backpropagation (Rumelhart et al., 1986) is a feed-forward network as shown in Figure 2a that updates the weights iteratively to map a set of input vectors $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p)$ to a set of corresponding output vectors $(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_p)$. The input \mathbf{x}_p corresponding to pattern or data point “ p ” is presented to the network and multiplied by the weights. All the weighted inputs to each unit of the upper layer are summed up, and produce an output governed by the following equations:

$$\mathbf{y}_p = f(\mathbf{W}_o \mathbf{h}_p + \boldsymbol{\theta}_o), \quad (2)$$

$$\mathbf{h}_p = f(\mathbf{W}_h \mathbf{x}_p + \boldsymbol{\theta}_h), \quad (3)$$

where \mathbf{W}_o and \mathbf{W}_h are the output and hidden layer weight matrices, \mathbf{h}_p is the vector denoting the response of hidden layer for pattern “ p ”, $\boldsymbol{\theta}_o$ and $\boldsymbol{\theta}_h$ are the output and hidden layer bias vectors, respectively and $f(\cdot)$ is the sigmoid activation function. The cost function to be minimized in standard Backpropagation is the sum of squared error defined as:

$$E = \frac{1}{2} \sum_p (\mathbf{t}_p - \mathbf{y}_p)^T (\mathbf{t}_p - \mathbf{y}_p) \quad (4)$$

where \mathbf{t}_p is the target output vector for pattern “ p ”. The algorithm uses *gradient descent technique* to adjust the connection weights between neurons. Denoting the fan-in weights to a single neuron by a weight vector \mathbf{w} , its update in the t -th epoch is governed by the following equation:

$$\Delta \mathbf{w}_t = -\eta \nabla E(\mathbf{w})|_{\mathbf{w} = \mathbf{w}^{(t)}} + \alpha \Delta \mathbf{w}_{t-1} \quad (5)$$

The parameters η and α are the learning rate and the momentum factor, respectively. The learning rate parameter controls the step size in each iteration. For a large-scale problem, Backpropagation learns very slowly and its convergence largely depends on choosing suitable values of η and α by the user.

Scaled Conjugate Gradient Algorithm

The error surface in Backpropagation may contain long ravines with sharp curvature and a gently sloping floor, which causes slow convergence. In conjugate gradient methods, a search is performed along *conjugate directions*, which produces generally faster convergence than *steepest descent directions* (Hagan, Demuth, & Beale, 1996). In steepest descent search, a new direction is perpendicular to the old direction. This approach to the minimum is a zigzag path and one step can be mostly undone by the next. In conjugate gradient methods, a new search direction spoils as little as possible the minimization achieved by the previous direction and the step size is adjusted in each iteration. The general procedure to determine the new search direction is to combine the new steepest descent direction with the previous search direction so that the current and previous search directions are conjugate. Conjugate gradient techniques are based on the assumption that, for a general nonquadratic error function, error in the neighborhood of a given point is locally quadratic. The weight changes in successive steps are given by the following equations:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \mathbf{d}_t \quad (6)$$

$$\mathbf{d}_t = -\mathbf{g}_t + \beta_t \mathbf{d}_{t-1} \quad (7)$$

with

$$\mathbf{g}_t \equiv \nabla E(\mathbf{w})|_{\mathbf{w} = \mathbf{w}_t} \quad (8)$$

$$\beta_t = \frac{\mathbf{g}_t^T \mathbf{g}_t}{\mathbf{g}_{t-1}^T \mathbf{g}_{t-1}} \quad \text{or} \quad \beta_t = \frac{\Delta \mathbf{g}_{t-1}^T \mathbf{g}_t}{\mathbf{g}_{t-1}^T \mathbf{d}_{t-1}} \quad \text{or} \quad \beta_t = \frac{\Delta \mathbf{g}_{t-1}^T \mathbf{g}_t}{\mathbf{g}_{t-1}^T \mathbf{g}_{t-1}} \quad (9)$$

where \mathbf{d}_t and \mathbf{d}_{t-1} are the conjugate directions in successive iterations. The step size is governed by the coefficient α_t , and the search direction is determined by β_t . In scaled conjugate gradient, the step size α_t is calculated by the following equations:

$$\alpha_t = -\frac{\mathbf{d}_t^T \mathbf{g}_t}{\delta_t} \quad (10)$$

$$\delta_t = \mathbf{d}_t^T \mathbf{H}_t \mathbf{d}_t + \lambda_t \|\mathbf{d}_t\|^2 \quad (11)$$

where λ_t is the scaling coefficient and \mathbf{H}_t is the Hessian matrix at iteration t . λ is introduced because, in case of nonquadratic error function, the Hessian matrix need not be positive definite. In this case, without λ , δ may become negative and a weight update may lead to an increase in error function. With sufficiently large λ , the modified Hessian is guaranteed to be positive ($\delta > 0$). However, for large values of λ , step size will be smaller. If the error function is not quadratic or $\delta < 0$, λ can be increased to make $\delta > 0$. In case of $\delta < 0$, Moller (1993) suggested the appropriate scale coefficient $\bar{\lambda}_t$ to be:

$$\bar{\lambda}_t = 2 \left(\lambda_t - \frac{\delta_t}{\|\mathbf{d}_t\|^2} \right) \quad (12)$$

Rescaled value $\bar{\delta}_t$ of δ_t is then be expressed as:

$$\bar{\delta}_t = \delta_t + (\bar{\lambda}_t - \lambda_t) \|\mathbf{d}_t\|^2 \quad (13)$$

The scaled coefficient also needs adjustment to validate the local quadratic approximation. The measure of quadratic approximation accuracy, Δ_t , is expressed by:

$$\Delta_t = \frac{2\{E(\mathbf{w}_t) - E(\mathbf{w}_t + \alpha_t \mathbf{d}_t)\}}{\alpha_t \mathbf{d}_t^T \mathbf{g}_t} \quad (14)$$

If Δ_t is close to 1, then the approximation is a good one and the value of λ_t can be decreased (Bishop, 1995). On the contrary, if Δ_t is small, the value of λ_t has to be increased. Some prescribed values suggested in Moller (1993) are as follows:

$$\begin{aligned} \text{For } \Delta_t > 0.75, & \quad \lambda_{t+1} = \lambda_t / 2 \\ \text{For } \Delta_t < 0.25, & \quad \lambda_{t+1} = 4\lambda_t \\ \text{Otherwise,} & \quad \lambda_{t+1} = \lambda_t \end{aligned}$$

Bayesian Regularization Algorithm

A desired neural network model should produce small error not only on sample data but also on out of sample data. To produce a network with better generalization ability,

MacKay (1992) proposed a method to constrain the size of network parameters by regularization. Regularization technique forces the network to settle to a set of weights and biases having smaller values. This causes the network response to be smoother and less likely to overfit (Hagan et al., 1996) and capture noise. In regularization technique, the cost function F is defined as:

$$F = \gamma E_D + (1 - \gamma) E_w \quad (15)$$

where E_D is the same as E defined in Equation 4, $E_w = \|\mathbf{w}\|^2 / 2$ is the sum of squares of the network parameters, and $\gamma (< 1.0)$ is the performance ratio parameter, the magnitude of which dictates the emphasis of the training on regularization. A large γ will drive the error E_D to small value whereas a small γ will emphasize parameter size reduction at the expense of error and yield smoother network response. One approach of determining optimum regularization parameter automatically is the Bayesian framework (Mackay, 1992). It considers a probability distribution over the weight space, representing the relative degrees of belief in different values for the weights. The weight space is initially assigned some prior distribution. Let $D = \{\mathbf{x}_m, \mathbf{t}_m\}$ be the data set of the input-target pair, m being a label running over the pair and M be a particular neural network model. After the data is taken, the posterior-probability distribution for the weight $p(\mathbf{w}|D, \gamma, M)$ is given according to the Bayesian rule.

$$p(\mathbf{w} | D, \gamma, M) = \frac{p(D | \mathbf{w}, \gamma, M) p(\mathbf{w} | \gamma, M)}{p(D | \gamma, M)} \quad (16)$$

where $p(\mathbf{w}|\gamma, M)$ is the prior distribution, $p(D|\mathbf{w}, \gamma, M)$ is the likelihood function, and $p(D|\gamma, M)$ is a normalization factor. In Bayesian framework, the optimal weight should maximize the posterior probability $p(\mathbf{w}|D, \gamma, M)$, which is equivalent to maximizing the function in Equation 15. Applying the Bayes' rule optimizes the performance ratio parameter.

$$p(\gamma | D, M) = \frac{p(D | \gamma, M) p(\gamma | M)}{p(D | M)} \quad (17)$$

If we assume a uniform prior distribution $p(\gamma|M)$ for the regularization parameter γ , then maximizing the posterior probability is achieved by maximizing the likelihood function $p(D|\gamma, M)$. Since all probabilities have a Gaussian form it can be expressed as:

$$p(D | \gamma, M) = (\pi / \gamma)^{-N/2} [\pi / (1 - \gamma)]^{-L/2} Z_F(\gamma) \quad (18)$$

where L is the total number of parameters in the neural network (NN). Supposing that F has a single minimum as a function of \mathbf{w} at \mathbf{w}^* and has the shape of a quadratic function in a small area surrounding that point, Z_F is approximated as (Mackay, 1992):

$$Z_F \approx (2\pi)^{L/2} \det^{-1/2} H^* \exp(-F(\mathbf{w}^*)) \quad (19)$$

where $H = \gamma \nabla^2 E_D + (1-\gamma) \nabla^2 E_w$ is the Hessian matrix of the objective function. Using Equation 19 in Equation 18, the optimum value of γ at the minimum point can be determined.

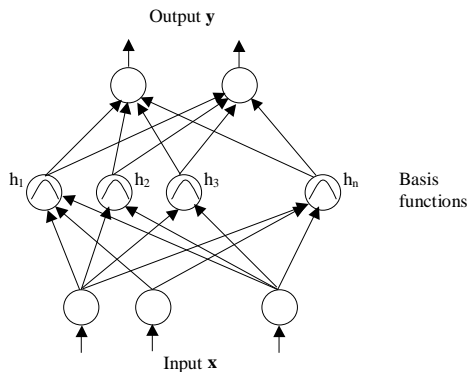
Foresee and Hagan (1997) proposed to apply the Gauss-Newton approximation to the Hessian matrix, which can be conveniently implemented if the Levenberg-Marquart optimization algorithm (More, 1977) is used to locate the minimum point. This minimizes the additional computation required for regularization.

Radial Basis Function Neural Network

Figure 3 shows a radial basis function neural network (RBFNN). A radial-basis-function network has a hidden layer of radial units and a linear-output layer units. Similar to biological receptor fields, RBFNNs employ local receptor fields to perform function mappings. Unlike hidden layer units in preceding algorithms where the activation level of a unit is determined using weighted sum, a radial unit (i.e., local receptor field) is defined by its center point and a radius. The activation level of the i -th radial unit is:

$$h_i = R_i(\mathbf{x}) = R_i(\|\mathbf{x} - \mathbf{u}_i\| / \sigma_i) \quad (20)$$

Figure 3. A radial-basis-function neural network (Note: not all the interconnections are shown; each basis function acts like a hidden unit.)



where \mathbf{x} is the input vector, \mathbf{u}_i is a vector with the same dimension as \mathbf{x} denoting the center, σ is width of the function and $R_i(\cdot)$ is the i -th radial basis function. Typically $R(\cdot)$ is a Gaussian function:

$$R_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{u}_i\|^2}{2\sigma_i^2}\right) \quad (21)$$

or a logistic function:

$$R_i(\mathbf{x}) = \frac{1}{1 + \exp(\|\mathbf{x} - \mathbf{u}_i\|^2 / \sigma_i^2)} \quad (22)$$

The activation level of radial basis function h_i for i -th radial unit is at its maximum when \mathbf{x} is at the center \mathbf{u}_i of that unit. The i -th component of the final output \mathbf{y} of a RBFNN can be computed as the weighted sum of the outputs of the radial units as:

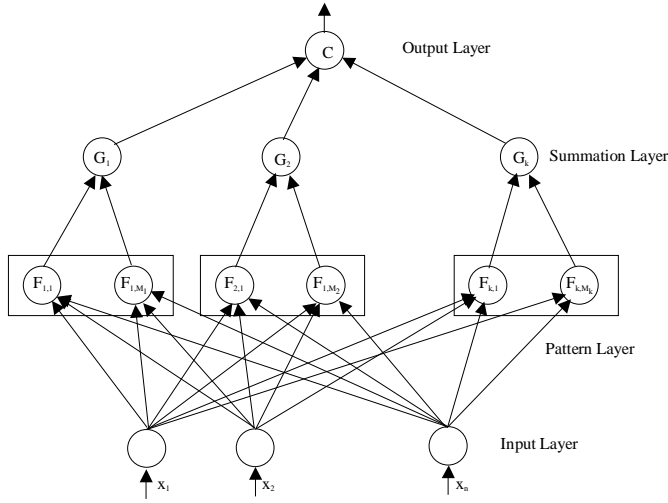
$$y_i = \sum_i \omega_i R_i(\mathbf{x}) \quad (23)$$

where ω_i is the connection weight between the radial unit i and the output unit, and the solution can be written directly as $\mathbf{w}^t = \mathbf{R}^t \mathbf{y}$, where \mathbf{R} is a vector whose components are the output of radial units and \mathbf{y} is the target vector. The adjustable parameters of the network, that is, the center and shape of radial basis units (\mathbf{u}_i , σ_i and ω_i) can be trained by a supervised training algorithm. Centers should be assigned to reflect the natural clustering of the data. Lowe (1995) proposed a method to determine the centers based on standard deviations of training data. Moody and Darken (1989) selected the centers by means of data clustering techniques like k -means clustering and σ 's are then estimated by taking the average distance to the several nearest neighbors of \mathbf{u}_i 's. Nowlan and Hinton (1992) proposed soft competition among radial units based on maximum likelihood estimation of the centers.

Probabilistic Neural Network

In case of classification problem neural network learning can be thought of estimating the probability density function (pdf) from the data. In regression task, the output of the network can be regarded as the expected value of the model at a given point in input space. An alternative approach to pdf estimation is the kernel-based approximation and this

Figure 4. A probabilistic neural network



motivates two types of networks that are similar to radial-basis-function networks: (a) probabilistic neural network (PNN) designed for classification task and (b) generalized regression neural network (GRNN). Specht (1990) introduced the PNN. It is a supervised NN that is widely used in the area of pattern recognition, nonlinear mapping, and estimation of the probability of class membership and likelihood ratios (Specht & Romsdahl, 1994). It is also closely related to the Bayes classification rule, and Parzen nonparametric probability density function estimation theory (Parzen, 1962; Specht, 1990). The fact that PNNs offer a way to interpret the network's structure in terms of probability-density functions is an important merit of this type of network. PNNs also achieve faster training than Backpropagation type feedforward neural networks.

The structure of an PNN is similar to that of feedforward NNs, although the architecture of an PNN is limited to four layers: the *input layer*, the *pattern layer*, the *summation layer*, and the *output layer*, as illustrated in Figure 4. An input vector \mathbf{x} is applied to the n input neurons and is passed to the pattern layer. The neurons of the pattern layer are divided into K groups, one for each class. The i -th pattern neuron in the k -th group computes its output using a Gaussian kernel of the form:

$$F_{k,i}(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_{k,i}\|^2}{2\sigma^2}\right) \quad (24)$$

where $\mathbf{x}_{k,i}$ is the center of the kernel, and σ , called the spread (smoothing) parameter, determines the size of the receptive field of the kernel. The summation layer contains one

neuron for each class. The summation layer of the network computes the approximation of the conditional class probability functions through a combination of the previously computed densities as per the following equation:

$$G_k(\mathbf{x}) = \sum_{i=1}^{M_k} \omega_{ki} F_{ki}(\mathbf{x}), \quad k \in \{1, \dots, K\}, \quad (25)$$

where M_k is the number of pattern neurons of class k , and ω_{ki} are positive coefficients satisfying, $\sum_{i=1}^{M_k} \omega_{ki} = 1$. Pattern vector \mathbf{x} belongs to the class that corresponds to the summation unit with the maximum output.

The parameter that needs to be determined for an optimal PNN is the smoothing parameter. One way of determining this parameter is to select an arbitrary set of σ , train the network, and test on the validation set. The procedure is repeated to find the set of σ that produces the least misclassification. An alternative way to search the optimal smoothing parameter was proposed by Masters (1995). The main disadvantages of a PNN algorithm is that the network can grow very big and become slow, especially when executing a large training set, making it impractical for a large classification problem.

Generalized Regression Neural Network

As mentioned earlier, a generalized regression neural network (GRNN) is also based on radial basis function and operates in a similar way to PNN but performs regression instead of classification tasks. Like PNN, GRNN architecture is comprised of four layers: the input, pattern, summation, and output layers. An GRNN represents each training sample as a kernel and establishes a regression surface by using a Parzen-window estimator (Parzen, 1962) with all the kernel widths assumed to be identical and spherical in shape. Assuming the function to be approximated by $y = g(\mathbf{x})$ where $\mathbf{x} \in \mathfrak{R}^n$ is an independent variable vector and $y \in \mathfrak{R}$ is the dependent variable, regression in an GRNN is carried out by the expected conditional mean of y as shown in the following equation:

$$E[y | \mathbf{x}] = \frac{\int_{-\infty}^{\infty} yg(\mathbf{x}, y) dy}{\int_{-\infty}^{\infty} g(\mathbf{x}, y) dy} \quad (26)$$

where $g(\mathbf{x}, y)$ is the Parzen probability density estimator, $E[y|\mathbf{x}]$ is the expected value of y given \mathbf{x} . When value of $g(\mathbf{x}, y)$ is unknown, it can be estimated from a sample of observations of \mathbf{x} and y . For a nonparametric estimates of $g(\mathbf{x}, y)$, the class of consistent estimators proposed by Parzen (1962) and extended to the multidimensional case by Cacoullos (1966) is used. The predicted output produced by GRNN network is given by:

$$\hat{y}(\mathbf{x}) = \frac{\sum_i^m y_i \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma_i^2}\right)}{\sum_i^m \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma_i^2}\right)} \quad (27)$$

where (\mathbf{x}_i, y_i) represents the i -th sample and m is the number of training samples. One of the main drawbacks of the GRNN is the extensive computational resources necessary for processing kernels and optimizing its width. Different approaches were also proposed to reduce the number of kernels in the GRNN.

The unsupervised learning algorithms like Adaptive Resonance Theory (ART), Self Organizing Map (SOM) are not so commonly used in financial and manufacturing applications and hence left out of discussion for the current chapter. Interested readers may consult works by Carpenter and Grossberg (1988) and Kohonen (1998).

Neural Network Applications in Finance

One of the main applications of neural networks in finance is trading and financial forecasting. Successful applications of neural networks includes a wide range of real world problems, for example, future price estimation, derivative securities pricing and hedging, exchange rate forecasting, bankruptcy prediction, stock performance and selection, portfolio assignment and optimization, financial volatility assessment, and so on. Demonstrated results and novelty of neural network applications have attracted practitioners in this field. Some of these applications are briefly reviewed in the following section.

Bankruptcy Prediction

Bankruptcy prediction has been an important and widely studied topic. The prediction of the likelihood of failure of a company given a number of financial measures, how soon an “ill” business can be identified, possibility of identifying the factors that put a business at risk — these are of main interest in bank lending. Atiya (2001) and Vellido, Lisboa, and Vaughan (1999) conducted a survey on the use of neural networks in business applications that contains a list of works covering bankruptcy prediction. Supervised neural network models have been tested against a number of techniques, like discriminant analysis (Kiviluoto, 1998; Olmeda & Fernandez, 1997); regression (Fletcher & Goss, 1993; Leshno & Spector, 1996); decision trees (Tam & Kiang, 1992); k -nearest neighbor (Kiviluoto); multiple adaptive regression splines (MARS) (Olmeda & Fernandez); case-based reasoning (Jo, Han, & Lee, 1997), and so on. In most cases, neural network models attained significantly better accuracy compared to other methods.

Credit Scoring

Credit scoring is another area of finance where neural network applications have been explored. From a bank lending point of view, it is important to distinguish a good debtor from a bad debtor by assessing the credit risk factor of each applicant. It can be distinguished from the past behavioral or performance scoring in which the repayment behavior of an applicant is analyzed to make a credit decision. The availability of data in this field is rather restricted (Desay, Crook, & Overstreet, 1996). Hecht-Nielson Co. has developed a credit-scoring system that increased profitability by 27% by identifying good credit risk and poor credit risk (Harston, 1990). Glorfeld and Hardgrave (1996), Jagielska and Jaworski (1996), Leigh (1995), Piramuthu, Shaw, and Gentry (1994), Torsun (1996), among others, have also reported similar works on credit evaluation and scoring. A variety of data sizes, ranges of variables, and techniques to select appropriate variables were investigated in those studies.

Investment Portfolio

For every investment, there is a tradeoff between risk and return. So, it is necessary to ensure a balance between these two factors. Optimizing one's portfolio investment by analyzing those factors, maximizing the expected returns for a given risk, and rebalancing when needed is crucial for secure investment. Steiner and Wittkemper (1997) developed a portfolio structure optimization model on a day-to-day trading basis. While the stock decisions are derived from a nonlinear dynamic capital market model, the underlying estimation and forecast modules are based on the neural network model. Using German stock prices from 1990 to 1994, this model leads to a portfolio that outperforms the market portfolio by about 60%. Hung, Liang, and Liu (1996) proposed an integration of arbitrage pricing theory (APT) and an ANN to support portfolio management and report that the integrated model beats the benchmark and outperforms the traditional ARIMA model. Yeo, Smith, Willis, and Brooks (2002) also used k -means clustering and neural networks for optimal portfolio selection. Classification of policy holders into risk groups and predicting the claim cost of each group were done using k -means clustering while price sensitivity of each group was estimated by neural networks. Chapados and Bengio (2001) showed that a neural network-based asset allocation model can significantly outperform the benchmark market performance.

Foreign Currency Exchange Rates

Modeling foreign currency exchange rates is an important issue for the business community. The investment companies are dependent on the prediction of accurate exchange rates so that they may make investment decisions. This is quite a challenging job as the rates are inherently noisy, nonstationary, and deterministically chaotic. Yao & Tan (2000) developed a neural network model using six simple indicators to predict the exchange rate of six different currencies against the U.S. dollar. The ANN model

demonstrated superior performance in comparison with the ARIMA-based model. Kamruzzaman and Sarker (2003) used three different neural network learning algorithms to predict exchange rates and found that all algorithms performed better than traditional methods when compared against with five different performance metrics. Medeiros, Veiga, and Pedreira (2001) proposed a novel flexible model called neurocoefficient smooth transition autoregression (NCSTAR), an ANN to test for and model the nonlinearities in monthly exchange rates.

Stock Market Analysis

Stock analysis has long been one of the most important applications of neural networks in finance. Most international investment bankers and brokerage firms have major stakes in overseas markets. Hence, this topic has attracted considerable attentions from the research community. There have been numerous research articles related to this topic. These include works by Chiang, Urban, and Baldrige (1996), Kim and Chun (1998), and Teixeira and Rodrigues (1997) on stock market index prediction; Barr and Mani (1994) and Yoon, Guimaraes, and Swales (1994) on stock performance/selection prediction; Wittkemper and Steiner (1996) on stock risk prediction; and Brook (1998), Donaldson and Kamstra (1996), and Refenes and Holt (2001) on stock volatility prediction. In most cases, neural networks outperformed other statistical methods.

Other Applications

Other applications include detecting financial fraud; creating wealth; and modeling the relationship among corporate strategy, its financial status, and performance (Smith & Gupta, 2000). Holder (1995) reports that Visa International deployed a neural network-based fraud detection system that saved it an estimated \$40 million within the first 6 months of its operation.

Apart from theoretical research, Coakely and Brown (2000) describe a number of ANN-based systems that are widely used in commercial applications. These are:

- *FALCON*, used by six of the ten largest credit card companies to screen transactions for potential fraud.
- *Inspector*, used by Chemical Bank to screen foreign currency transactions.
- Several ANNs used to assist in managing investments by making predictions about debt and equity securities, as well as derivative instruments. Fadlalla and Lin (2001) cited examples from companies like Falcon Asset management, John Deere and Co., Hyman Beck and Company, Multiverse Systems, Advanced Investment Technology, and Ward System who used neural network-based systems. It has been reported that a significant number of Fortune-1000 companies use neural networks for financial modeling.

- Several ANNs used for credit granting, including GMAC's *Credit Adviser* that grants instant credit for automobile loans.
- *AREAS*, used for residential property valuation.

For other financial applications and more detailed survey, interested readers are referred to the articles by Coakley and Brown (2000), Fadlalla and Lin (2001), Smith and Gupta (2000), Vellido et al. (1999), and Wong, Lai, and Lam (2000).

Neural Network Applications in Manufacturing

In this section, a brief review of ANN applications in manufacturing design, planning, and operations will be presented. The overall applications can be classified as condition monitoring, cost estimation, fault diagnosis, parameter selection, production scheduling, manufacturing cell formation, quality control, and others.

Condition Monitoring

In manufacturing, condition monitoring is a major application area for ANNs. These applications involve monitoring different manufacturing operations and/or operational conditions such as tool wearing and breaking, metal forming, and drilling and machining accuracy.

The process of metal forming involves several dies and punches used progressively to form a part. Tooling is critical in metal forming, as continual tool replacement and maintenance reduces productivity, raises manufacturing cost, and increases defective item production. The ANN models, taking data from an online condition monitoring system, can predict tool life that would help to generate an appropriate maintenance schedule. Kong and Nahavandi (2002) developed such a model for the forging process that uses a multilayer error back propagation network. The inputs of the model were force, acoustic emission signals, process parameters (such as tool temperature, stroke rates, and surface lubrication condition of in-feed material), and expected life. The model helps to predict the tool condition, maintenance schedule, and tool replacement. Similar techniques can be applied to other metal forming processes.

Turning is a common metal cutting operation in manufacturing. In turning operations, flank wear on the cutting tool directly affects the work piece dimension and the surface quality. Choudhury et al. (1999) developed a methodology in which an optoelectronic sensor was used in conjunction with a multilayered neural network for predicting the flank wear. The digitized sensor signal, along with the cutting parameters, formed the inputs to a three-layer feedforward fully connected neural network. The neural network used a Backpropagation algorithm. Results showed the ability of the neural network to accurately predict the flank wear.

Huang and Chen (2000) developed an in-process tool breakage detection system using a neural network for an end mill operation. The inputs of the model were cutting force and

machining parameters such as spindle speed, feed rate, and depth of cut. The output was to detect the tool breakage conditions. Per their report, the neural networks were capable of detecting tool condition accurately. Wu (1992) developed a neural network model to detect tool failure based on the level of cutting force and vibration or acoustic emission.

Brophy et al. (2002) proposed a two-stage neural network model to detect anomalies in the drilling process. The network was used to classify drilling operations as *normal* or *abnormal* (e.g., tool breakage or missing tool). The network used spindle power signal (acquired over all or part of the operation) as the input. A limitation of the approach is that it requires the full signal before a classification is made.

Cost Estimation

The estimation of future production cost is a key factor in determining the overall performance of a new product's development and product redesigning process. Usually, the cost per unit of a given finished good is the sum of different resources such as raw materials, components, energy, machinery, and plants. The quantification of the use of each resource is extremely difficult. Cavalieri et al. (2004) proposed an ANN technique for the estimation of the unitary manufacturing costs of a new type of brake disks produced by an Italian manufacturing firm. The results seem to confirm the validity of the neural network theory in this application field, but not a clear superiority with respect to the traditional parametric approach. However, the ANN seems to be characterised by a better trade-off between precision and cost of development. Zhang, Fuh, and Chan (1996) illustrated the use of a neural network-based model for the estimation of the packaging cost based on the geometrical characteristics of the packaged product.

Fault Diagnosis

Identifying the cause of process abnormalities is important for process automation. Knapp and Wang (1992) studied the application of a Backpropagation network to fault diagnosis of a Computer Numerical Control (CNC) machine using vibration data. Knapp, Javadpour, and Wang (2000) presented a real-time neural network-based condition monitoring system for rotating mechanical equipment. There has been much effort recently in making a fusion of fuzzy logic and neural networks for better performance in decision making processes. The uncertainties involved in the input description and output decision are taken care of by the concept of fuzzy sets while the neural net theory helps in generating the required decision region. Javadpour and Knapp (2003) implemented a neural network model to diagnosis faults with high prediction accuracy in an automated manufacturing environment. Their model incorporated the fuzzy concept to capture uncertain input data.

Nondestructive testing for fault detection in welding technology is very expensive. However, the correct detection of welding faults is important to the successful detection of an automated welding inspection system. Liao, Triantaphyllou, and Chang (2003) investigated the performance of a multilayer perception neural networks in welding fault detection.

Parameter Selection

Machining-parameter selection is a crucial task in a manufacturing environment. In the conventional turning process, the parameters referred are cutting speed, feed rate, and depth of cut. The parameters play an important role in efficient utilization of machine tools and significantly influence the overall manufacturing cost. Wong and Hamouda (2003) used a feedforward neural network to predict optimum machining parameters under different machining conditions. Although they introduced a new type of artificial neuron, the Backpropagation algorithm was used to optimize the network component representation.

Raj et al. (2000) developed a Backpropagation neural network to estimate the cutting forces based on speed, feed, and depth of cut for machining a mild-steel specimen with a high-speed steel (HSS) tool. In addition, they modeled the effect of tool geometry (i.e., rake angle, cutting edge location, and orientation of tool face) on cutting forces. Viharos, Monostori, and Markos (1999) applied a neural network to predict the cutting forces based on cutting parameters for an expected surface roughness.

Wang (2004) proposed a hybrid two-phase neural network approach for modeling a manufacturing process under a lack of observations, which is designed for determining cutting parameters in wire Electrical Discharge Machining (EDM). A combination of ANN and genetic algorithms was also used for determining cutting parameters in machining operations (Cus & Balic, 2003) and manufacturing-process parameters (Li, Su, & Chiang, 2003).

Zuperl, Cus, Mursec, and Ploj (2004) proposed a new hybrid technique for optimal selection of cutting parameters. The approach uses 10 technological constraints and is based on the maximum production rate criteria. It describes the multiobjective optimization of cutting conditions by means of an ANN and a routine (known as OPTIS) by taking into consideration the technological, economic, and organizational limitations. The analytical module OPTIS selects the optimum cutting conditions from commercial databases with respect to minimum machining costs. By selection of optimum cutting conditions, it is possible to reach a favorable ratio between the low machining costs and high productivity taking into account the given limitation of the cutting process. Experimental results show that the proposed optimization algorithm for solving the nonlinear constrained programming (NCP) problems is both effective and efficient, and can be integrated into an intelligent manufacturing system for solving complex machining optimization problems.

Production Scheduling

Production scheduling is the allocation of resources over time to perform a collection of tasks. Of all kinds of production scheduling problems, the job shop scheduling is one of the most complicated problems. It aims to allocate m machines to perform n jobs in order to optimize certain criteria. Fonseca and Navarrese (2002) developed a feedforward multilayered neural network through the back error propagation learning algorithm to provide a versatile job shop scheduling analysis framework. Yang and Wang (2000)

proposed a constraint satisfaction adaptive neural network, together with several heuristics, to solve a generalized job shop scheduling problem. Their results demonstrated that the proposed neural network and its combined approaches are efficient with respect to the quality of solution and the solving speed.

Lee and Shaw (2000) considered a classical problem of sequencing a set of jobs that arrive in different combinations over time in a manufacturing flow shop. They developed a two-level neural network that incrementally learns sequencing knowledge. Based on the knowledge gained, the neural network makes real-time sequencing decisions for a set of jobs. Akyol (2004) used a neural network with six different heuristic algorithms for flow-shop scheduling.

Manufacturing Cell Formation and Related Problems

The design of a manufacturing cell refers to obtaining good performance measures, such as using an optimal number of resources, predefined utilization rates, minimizing the production time, and so on. Christodoulou and Gaganis (1998) presented a neural network approach in determining the appropriate manufacturing cell configuration that meets the required performance measures.

Cell formation is a key issue in implementing cellular manufacturing and consists of decomposing the shop in distinct manufacturing cells, each one dedicated to the processing of a family of similar part types. Guerrero, Lozano, Smith, Canca, and Kwok (2002) proposed a methodology for cell formation in which a self-organizing neural network is used to compute weighted similarity coefficients and cluster parts. Then a linear network flow model is used to assign machines to families. Moon and Chi (1992) used a neural network model to solve the part family formation problem. They combined neural network technique with the flexibility of the similarity coefficient method. Manufacturing information such as the sequence of operations, lot size, and multiple process plans were given special consideration in their approach to solve a generalized part-family formation problem. Further neural network applications can be found for the part-machine grouping problem (Kaparathi & Suresh, 1992), part family formation (Lee, Malave, & Ramachandran, 1992), and machine cell identification (Lee et al., 1992).

Quality Control

Product quality refers to form errors, surface finish and dimensional errors produced on components during machining. Several factors such as machine accuracy, tool/work deflections, process conditions, and so on, dictate the product accuracy produced during manufacturing. Despite significant research work, there is no integrated product quality model reported to predict the product quality performance in CNC machining. Suneel, Pande, and Date (2002) reported the development of an intelligent product quality model for CNC turning using neural network techniques.

Reliable and accurate quality control is an important element in textile manufacturing. For many textile products, a major quality control requirement is judging seam quality.

Bahlmann et al. (1999) presented a method for an automated quality control of textile seams, which is aimed to establish a standardized quality measure and to lower cost in manufacturing. The system consists of a suitable image acquisition setup, an algorithm for locating the seam, a feature extraction stage, and a neural network of the self-organizing map type for feature classification.

Other Applications

Applications of neural network in other problems such as industrial pattern recognition (Yao, Freeman, Burke, & Yang, 1991), identification of appropriate decision criteria (Chryssolouris, Lee, & Domroese, 1991), agile and flexible manufacturing (Shimizu, Tanaka, & Kawada, 2004) and economic order quantity (Ziarati, 2000) have also been reported in the literature. To differentiate the pattern recognition problems from other disciplines, we would like to mention here that this problem in this context is to recognize industrial crews, bolts, and so on.

Conclusion

Artificial neural networks possess many desirable features that have made them suitable for practical financial and manufacturing applications. In this chapter, we have provided a brief description of ANN architectures and different learning algorithms that are most commonly used in these applications. Interested readers are also directed to more detailed descriptions of the algorithms for their relative advantages and disadvantages for further information. Specific areas in finance and manufacturing that have experienced remarkable results by modeling with neural networks are described and some of the important and relevant works are reported. The subsequent chapters of this book will present some of the recent developments of ANN applications in finance and manufacturing, and discuss various modeling issues.

References

- Abu-Mostafa, Y. S. (2001). Financial model calibration using consistency hints. *IEEE Transaction on Neural Networks*, 12(4), 791-808.
- Akyol, D. E. (2004). Applications of neural networks to heuristic scheduling algorithms. *Computers and Industrial Engineering*, 46, 679-696.
- Arai, M. (1989). Mapping abilities of three layer neural networks. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, Vol. 1 (pp. 419-423).

- Atiya, A. F. (2001). Bankruptcy prediction for credit risk using neural networks: A survey and new results. *IEEE Transactions on Neural Networks*, 12(4), 929-935.
- Bahlmann, C., Heidemann, G., & Ritter, H. (1999). Artificial neural networks for automated quality control of textile seams. *Pattern Recognition*, 32(6), 1049-1060.
- Barr, D. S., & Mani, G. (1994). Using neural nets to manage investments. *AI Expert*, 9, 6-21.
- Becraft, W. R., & Lee, P. L. (1993). An integrated neural network/expert system approach for fault diagnosis. *Computers and Chemical Engineering*, 17(10), 1001-1014.
- Bishop, C. M. (1995). *Neural networks for pattern recognition*. New York: Oxford University Press.
- Brooks, C. (1998). Predicting stock index volatility: Can market volume help? *Journal of Forecasting*, 17, 59-80.
- Brophy, B., Kelly, K., & Byrne, G. (2002). AI-based condition monitoring of the drilling process. *Journal of Materials Processing Technology*, 124, 305-310.
- Cacoullos, T. (1966). Estimation of multivariate density. *Annals of the Institute of Statistical Mathematics*, 18(2), 179-189.
- Carpenter, G. A., & Grossberg, S. (1988). The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3), 77-88.
- Cavalieri, S., Maccarrone, P., & Pinto, R. (2004). Parametric vs neural network models for the estimation of production costs: A case study in the automotive industry. *International Journal of Production Economics*, 91, 165-177.
- Chapados, N., & Bengio, M. (2001). Cost functions and model combination for VaR-based asset allocation using neural networks. *IEEE Transactions on Neural Networks*, 12(4), 890-907.
- Chiang, W., Urban, T. L., & Baldrige, G. W. (1996). A neural network approach to mutual fund net asset value forecasting. *International Journal of Management Science*, 24(2), 205-215.
- Cho, J. R., Shin, S. W., & Yoo, W. S. (2005). Crown shape optimization for enhancing tire wear performance by ANN. *Computers & Structures*, 83, 12-13, 920-933.
- Choudhury, S. K., Jain, V. K., & Rama Rao, Ch. V. V. (1999). On-line monitoring of tool wear in turning using a neural network. *International Journal of Machine Tools and Manufacture*, 39, 489-504.
- Christodoulou, M., & Gaganis, V. (1998). Neural network in manufacturing cell design. *Computers in Industry*, 36, 133-138.
- Chrysolouris, G., Lee, M., & Domroese, M. (1991). The use of neural networks in determining operational policies for manufacturing systems. *Journal of Manufacturing Systems*, 10(2), 166-175.
- Coakley, J., & Brown, C. (2000). Artificial neural networks in accounting and finance: Modeling issues. *International Journal of Intelligent Systems in Accounting, Finance & Management*, 9, 119-144.
- Cus, F., & Balic, J. (2003). Optimization of cutting process by GA approach. *Robotics and Computer Integrated Manufacturing*, 19, 113-121.

- Desay, V. S., Crook, J. N., & Overstreet, G. A., Jr. (1996). A comparison of neural networks and linear scoring models in the credit union environment. *European Journal of Operational Research*, 95, 24-37.
- Donaldson, R. G., & Kamstra, M. (1996). Forecast combining with neural networks. *Journal of Forecasting*, 15, 49-61.
- Durantou, M. (1996). Image processing by neural networks. *IEEE Micro*, 16(5), 12-19.
- Fadlalla, A., & Lin, C. H. (2001). An analysis of the applications of neural networks in finance. *Interfaces*, 31(4), 112-122.
- Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*, 2, 524-532.
- Fletcher, D., & Goss, E. (1993). Forecasting with neural networks: An application using bankruptcy data. *Information & Management*, 24, 159-167.
- Fonseca, D., & Navarrese, D. (2002). Artificial neural network for job shop simulation. *Advanced Engineering Informatics*, 16, 241-246.
- Foresee, F. D., & Hagan, M. T. (1997). Gauss-Newton approximation to Bayesian regularization. *International Joint Conference Neural Network*, 1930-1935.
- Fukumi, M., Omatu, S., & Nishikawa, Y. (1997). Rotation-invariant neural pattern recognition system estimating a rotation angle. *IEEE Transaction on Neural Networks*, 8(3), 568-581.
- Glorfeld, L. W. (1996). A methodology for simplification and interpretation of backpropagation-based neural network models. *Expert Systems with Applications*, 10(1), 37-54.
- Glorfeld, L. W., Hardgrave, B. C. (1996). An improved method for developing neural networks: The case of evaluating commercial loan credit worthiness. *Computers & Operations Research*, 23, 933-944.
- Guerrero, F., Lozano, S., Smith, K., Canca, D., & Kwok, T. (2002). Manufacturing cell formation using a new self-organizing neural network. *Computers & Industrial Engineering*, 42, 377-382.
- Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural network design*. Boston: PWS Publishing.
- Harston, C. T. (1990). Business with neural networks. In A. Maren, C. Harston, & R. Pap, (Eds.), *Handbook of neural computing applications*. CA: Academic Press.
- Holder, V. (1995, February 7). War on suspicious payment. *Financial Times*.
- Huang, P. T., & Chen, J. C. (2000). Neural network-based tool breakage monitoring system for end milling operations. *Journal of Industrial Technology*, 16(2), 2-7.
- Hung, S. Y., Liang, T. P., & Liu, V. W. (1996). Integrating arbitrage pricing theory and artificial neural networks to support portfolio management. *Decision Support Systems*, 18, 301-316.
- Jagielska, I., & Jaworski, J. (1996). Neural network for predicting the performance of credit card accounts. *Computational Economics*, 9(1), 77-82.

- Javadpour, R., & Knapp, G. M. (2003). A fuzzy neural network approach to machine condition monitoring. *Computers & Industrial Engineering*, 45, 323-330.
- Jin, X., Cheu, R. L., & Srinivasan, D. (2002). Development and adaptation of constructive probabilistics neural network in freeway incident detection. *Transportation Research Part C*, 10, 121-147.
- Jo, H., Han, I., & Lee, H. (1997). Bankruptcy prediction using case-based reasoning, neural network and discriminate analysis. *Expert Systems with Applications*, 13(2), 97-108.
- Kamruzzaman, J., & Sarker, R. (2003). Forecasting of currency exchange rates using ANN: A case study. In *Proceedings of the IEEE International Conference on Neural Network & Signal Processing* (pp. 793-797).
- Kaparthi, S., & Suresh, N. C. (1992). Machine-component cell formation in group technology: A neural network approach. *International Journal of Production Research*, 30(6), 1353-1367.
- Kim, S. H., & Chun, S. H. (1998). Graded forecasting using an array of bipolar predictions: Application of probabilistic neural networks to a stock market index. *International Journal of Forecasting*, 14, 323-337.
- Kiviluoto, K. (1998). Predicting bankruptcies with the self-organizing map. *Neurocomputing*, 21 (1-3), 203-224.
- Knapp, G. M., Javadpour, R., & Wang, H. P. (2000). An ARTMAP neural network-based machine condition monitoring system. *Journal of Quality in Maintenance Engineering*, 6(2), 86-105.
- Knapp, G. M., & Wang, H. P. (1992). Machine fault classification. A neural network approach. *International Journal of Production Research*, 30(4), 811-823.
- Kohonen, T. (1988). *Self-organisation and associative memory*. New York: Springer-Verlag.
- Kong, L. X., & Nahavandi, S. (2002). On-line tool condition monitoring and control system in forging processes. *Journal of Materials Processing Technology*, 125-126, 464-470.
- Kruschke, J. K. (1989). Improving generalization in backpropagation networks with distributed bottlenecks. *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, 1, 443-447.
- Kung, S. Y., & Hwang, J. N. (1988). An algebraic projection analysis for optimal hidden units size and learning rate in backpropagation learning. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, Vol. 1 (pp. 363-370).
- Lee, C. W. (1997). Training feedforward neural networks: An algorithm for improving generalization. *Neural Networks*, 10, 61-68.
- Lee, H., Malave, C. O., & Ramachandran, S. (1992). A self-organizing neural network approach for the design of cellular manufacturing systems. *Journal of Intelligent Manufacturing*, 325-332.

- Lee, I., & Shaw, M. J. (2000). A neural-net approach to real time flow-shop sequencing. *Computers & Industrial Engineering*, 38, 125-147.
- Lehtokangas, M. (2000). Modified cascade-correlation learning for classification. *IEEE Transactions on Neural Networks*, 11, 795-798.
- Leigh, D. (1995). Neural networks for credit scoring. In S. Goonatilake, & P. Treleaven (Eds.), *Intelligent systems for finance & business* (pp. 61-69). Chichester: Wiley.
- Leshno, M., & Spector, Y. (1996). Neural network prediction analysis: The bankruptcy case. *Neurocomputing*, 10, 125-147.
- Li, T. S., Su, C. T., & Chiang, T. L. (2003). Applying robust multi-response quality engineering for parameter selection using a novel neural-genetic algorithm. *Computers in Industry*, 50, 113-122.
- Liao, T. W., Triantaphyllou, E., & Chang, P. C. (2003). Detection of welding flaws with mlp neural network and case base reasoning. *International Journal of Intelligent Automation and Soft Computing*, 9(4), 259-267.
- Looney, C. G. (1996). Advances in feedforward neural networks: Demystifying knowledge acquiring black boxes. *IEEE Transactions on Knowledge & Data Engineering*, 8, 211-226.
- Lowe, D. (1995). Radial basis function networks. In M. A. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press.
- MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation*, 4, 415-447.
- Marinai, S., Gori, M., & Soda, G. (2005). Artificial neural networks for document analysis and recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(1), 23-35.
- Masters, T. (1995). *Advanced algorithms for neural networks*. New York: Wiley.
- Medeiros, M. C., Veiga, A., & Pedreira, C. E. (2001). Modelling exchange rates: Smooth transitions, neural networks, and linear models. *IEEE Transactions on Neural Networks*, 12(4), 755-764.
- Moller, A. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6, 525-533.
- Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2), 281-294.
- Moon, Y. B., & Chi, S. C. (1992). Generalized part family formation using neural network techniques. *Journal of Manufacturing Systems*, 11(3), 149-159.
- More, J. J. (1997). The Levenberg-Marquart algorithm: Implementation and theory: G. A. Watson (Ed.), *Numerical analysis. Lecture Notes in Mathematics*, 630, 105-116. London: Springer-Verlag.
- Nazeran, H., & Behbehani, K. (2000). Neural networks in processing and analysis of biomedical signals. In M. Akay (Ed.), *Nonlinear biomedical signal processing: Fuzzy logic, neural networks and new algorithms*. New York: IEEE Press.
- Nowlan, S. J., & Hinton, G. E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4), 473-493.

- Olmeda, I., & Fernandez, E. (1997). Hybrid classifiers for financial multicriteria decision making: The case of bankruptcy prediction. *Computational Economics*, 10(4), 317-335.
- Parzen, E. (1962). On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 3, 1065-1076.
- Piramuthu, S., Shaw, M. J., Gentry, J. A. (1994). A classification approach using multi-layered neural networks. *Decision Support Systems*, 11, 509-525.
- Raj, K. H., Sharma, R. S., Srivastava, S., & Patvardham, C. (2000). Modeling of manufacturing processes with ANNs for intelligent manufacturing. *International Journal of Machine Tools & Manufacture*, 40, 851-868.
- Refenes, A. P. N., & Holt, W. T. (2001). Forecasting volatility with neural regression: A contribution to model adequacy. *IEEE Transactions on Neural Networks*, 12(4), 850-865.
- Rumelhart, D. E., McClelland, J. L., & the PDP research group (1986). *Parallel Distributed Processing, 1*. MIT Press.
- Shimizu, Y., Tanaka, Y., & Kawada, A. (2004). Multi-objective optimization system, MOON² on the Internet. *Computers & Chemical Engineering*, 28, 821-828.
- Smith, K., & Gupta J. (2000). Neural networks in business: Techniques and applications for the operations researcher. *Computer & Operation Research*, 27, 1023-1044.
- Specht, D. F. (1990). Probabilistic neural networks. *Neural Networks*, 1(13), 109-118.
- Specht, D. F., & Romsdahl, H. (1994). Experience with adaptive probabilistic neural network and adaptive general regression neural network. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, Vol. 2 (pp. 1203-1208).
- Steiner, M., & Wittkemper, H. G. (1997). Portfolio optimization with a neural network implementation of the coherent market hypothesis. *European Journal of Operational Research*, 100, 27-40.
- Suneel, T. S., Pande, S. S., & Date, P. P. (2002). A technical note on integrated product quality model using artificial neural networks. *Journal of Materials Processing Technology*, 121(1), 77-86.
- Tam, K. Y., & Kiang, M. Y. (1992). Managerial applications of the neural networks: The case of bank failure predictions. *Management Science*, 38(7), 926-947.
- Teixeira, J. C., & Rodrigues, A. J. (1997). An applied study on recursive estimation methods, neural networks and forecasting. *European Journal of Operational Research*, 101, 406-417.
- Torsun, I. S. (1996). A neural network for a loan application scoring system. *The New Review of Applied Expert Systems*, 2, 47-62.
- Vellido, A., Lisboa, P., & Vaughan, J. (1999). Neural networks in business: A survey of applications (1992-1998). *Expert System with Applications*, 17, 51-70.
- Viharos, Z. J., Monostori, L., & Markos, S. (1999). Selection of input and output variables of ANN based modelling of cutting processes. In *Proceedings of the Workshop on Supervising and Diagnostics of Machining Systems*, Poland (pp. 121-131).

- Wang, G. N. (2004). Two-phase reverse neural network approach for modeling a complicate manufacturing process with small size. *Neural Information Processing — Letters and Reviews*, 2(1), 1-9.
- Wittkemper, H. G., & Steiner, M. (1996). Using neural networks to forecast the systematic risk of stocks. *European Journal of Operational Research*, 90, 577-588.
- Wong, B., Lai, V., & Lam, J. (2000). A bibliography of neural network business applications research: 1994-1998. *Computer & Operation Research*, 27, 1045-1076.
- Wong, S. V., & Hamouda, A. M. S. (2003). Machinability data representation with artificial neural network. *Journal of Materials Processing Technology*, 138, 538-544.
- Wu, B. (1992). An introduction to neural networks and their applications in manufacturing. *Journal of Intelligent Manufacturing*, 3, 391-403.
- Yang, S., & Wang, D. (2000). Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Network*, 11(2), 474-486.
- Yao, J., & Tan, C. T. (2000). A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, 34, 79-98.
- Yao, Y., Freeman, W. J., Burke, B., & Yang, Q. (1991). Pattern recognition by a distributed neural network: An industrial application. *Neural Networks*, 4, 103-121.
- Yeo, A. C., Smith, K. A., Willis, R. J., & Brooks, M. (2002). A mathematical programming approach to optimize insurance premium pricing within a data mining framework. *Journal of Operation Research Society*, 53, 1197-1203.
- Yoon, Y., Guimaraes, T., & Swales, G. (1994). Integrating artificial neural networks with rule-based expert systems. *Decision Support Systems*, 11, 497-507.
- Zhang, Y. F., Fuh, J. Y., & Chan, W. T. (1996). Feature-based cost estimation for packaging products using neural networks. *Computers in Industry*, 32, 95-113.
- Zhenyuan W., Yilu L., & Griffin, P. J. (2000). Neural net and expert system diagnose transformer faults. *Computer Applications in Power*, 13(1), 50-55.
- Ziarati, M. (2000). *Improving the supply chain in the automotive industry using kaizen engineering*. MPhil transfer thesis, De Montfort University, UK.
- Zuperl, U., Cus, F., Mursec, B., & Ploj, T. (2004). A hybrid analytical-neural network approach to the determination of optimal cutting conditions. *Journal of Materials Processing Technology*, 82-90.

Chapter II

Simultaneous Evolution of Network Architectures and Connection Weights in Artificial Neural Networks

Ruhul A. Sarker, University of New South Wales, Australia

Hussein A. Abbass, University of New South Wales, Australia

Abstract

Artificial Neural Networks (ANNs) have become popular among researchers and practitioners for modeling complex real-world problems. One of the latest research areas in this field is evolving ANNs. In this chapter, we investigate the simultaneous evolution of network architectures and connection weights in ANNs. In simultaneous evolution, we use the well-known concept of multiobjective optimization and subsequently evolutionary multiobjective algorithms to evolve ANNs. The results are promising when compared with the traditional ANN algorithms. It is expected that this methodology would provide better solutions to many applications of ANNs.

Introduction

Feed-forward ANNs have found extensive acceptance in many disciplines for modeling complex real-world problems including the finance and manufacturing domains. An ANN is formed from a group of units, called neurons or processing elements, connected with arcs, called synapses or links, where each arc is associated with a weight representing the strength of the connection, and usually the nodes are organized in layers. Each neuron has an input equal to the weighted sum of the outputs of those neurons connected to it. The weighted sum of the inputs represents the activation of the neuron. The activation signal is passed through a transfer function to produce a single neuron's output. The transfer function introduces nonlinearity to the network. The behavior of a neural network is determined by the transfer functions, the learning rule by which arcs update their weights, and the architecture itself in terms of the number of connections and layers. Training is the process of adjusting the networks' weights to minimize the difference between the network output and the desired output on a suitable metric space. Once the network is trained, it can be tested by a new dataset.

As previously mentioned, the performance of a neural network for a given problem depends on the transfer function, network architecture, connection weights, inputs, and learning rule. The architecture of an ANN includes its topological structure, that is, connectivity and number of nodes in the network. The architectural design is crucial for successful application of ANNs because the architecture has a significant impact on the overall processing capabilities of the network. In most function-approximation problems, one hidden layer is sufficient to approximate continuous functions (Basheer, 2000; Hecht-Nielsen, 1990). Generally, two hidden layers may be necessary for learning functions with discontinuities (Hecht-Nielsen, 1990). The determination of the appropriate number of hidden layers and number of hidden nodes in each layer is one of the important tasks in ANN design. A network with too few hidden nodes would be incapable of differentiating between complex patterns, leading to a lower estimate of the actual trend. In contrast, if the network has too many hidden nodes it will follow the noise in the data due to overparameterization leading to poor generalization for test data (Basheer & Hajmeer, 2000). As the number of hidden nodes increases, training becomes excessively time-consuming.

The most popular approach to finding the optimal number of hidden nodes is by trial and error. Methods for network growing such as cascade correlation (Fahlman & Lebiere, 1990) and for network pruning such as optimal brain damage (LeCun, Denker, & Solla, 1990) have been used to overcome the unstructured and somehow unmethodical process for determining good network architecture. However, all these methods still suffer from their slow convergence and long training time. Nowadays, many researchers use evolutionary algorithms to find the appropriate network architecture by minimizing the output error (Kim & Han, 2000; Yao & Liu, 1998).

Weight training in ANNs is usually formulated as a minimization of an error function, such as the mean square error between target and actual outputs averaged over all examples (training data) by iteratively adjusting connection weights. Most training algorithms,

such as *Backpropagation* (BP) and conjugate gradient are based on gradient descent (Basheer & Hajmeer, 2000; Hertz, Krogh, & Palmer, 1991). Although BP has some successful applications, the algorithm often gets trapped in a local minimum of the error function and is incapable of finding a global minimum if the error function is multimodal and/or non-differentiable (Yao, 1999). To overcome this problem, one can use evolutionary algorithms for weight training. The application of evolutionary algorithms for weight training can be found in Kim and Han (2000) and Yao and Liu (1998).

As discussed previously, it is necessary to determine the appropriate network architecture and connection weights to get the best performance out of ANNs. Most researchers treat network architecture and connection weights as two independent optimization problems. As Yao (1999) indicated, connection weights have to be learned after a near-optimal architecture is found. This is especially true if one uses an indirect encoding scheme, such as the developmental rule method. One major problem with the determination of architectures is noisy fitness evaluation (Yao & Liu, 1997). In order to reduce such noise, an architecture usually has to be trained many times from different random initial weights. This method dramatically increases the computational time for fitness evaluation. If we look at the theoretical side of such optimization problems, this sequential optimization procedure (first architecture optimization followed by weight optimization) will usually provide a suboptimal solution for the overall problem.

To overcome this problem, the natural choice is to determine the architecture and connection weights simultaneously by solving a single optimization problem with two objectives. Many researchers attempted to ignore the architecture and minimize only the mean sum square error function (Kim & Han, 2000; Yao & Liu, 1998). A comprehensive list of papers on this topic can be found in Yao (1999). However, if the contribution to the objective function of a subproblem is very low compared to the other, the effect of the first subproblem will not be reflected properly in the overall optimal solution. In such situations, simultaneous multiobjective optimization would be a better choice.

The purpose of this research is to determine the network architecture and connection weights of ANNs simultaneously by treating the problem as a multiobjective optimization problem. We believe the simultaneous evolution of architectures and connection weights in ANNs using the concept of multiobjective optimization will add a new direction of research in ANNs. In addition, we will show experimentally that this approach performs better than BP with much lower computational cost.

The chapter is organized as follows. After introducing the research context, multiobjective optimization and evolutionary algorithms are introduced in the next section followed by the proposed algorithm for simultaneous evolution of network architecture and connection weights. . Experimental results are presented in the “Experiments” section. Applications of the proposed algorithms to finance and manufacturing are discussed in the penultimate section, and conclusions are drawn in the final section.

Multiobjective Optimization and Evolutionary Algorithms

In this section, we briefly discuss multiobjective optimization, evolutionary algorithms, and the use of evolutionary algorithms for solving multiobjective optimization problems.

Multiobjective Optimization

Consider a multiobjective optimization model as presented next.

Objective Function	$f(\mathbf{x})$
Subject to	$\mathbf{x} \in X$

Where \mathbf{x} is a vector of decision variables (x_1, x_2, \dots, x_k) , f is a vector objective function with components f_1, \dots, f_n . Here f_1, \dots, f_n , are functions on E_n and X is a nonempty set in E_n . The set X represents simple constraints that could be easily handled explicitly, such as lower and upper bounds on the variables.

We wish to determine the optimal \mathbf{x} , which optimizes f , satisfying the variable bounds. In the vector objective function, the type of optimization of individual function could be maximization, minimization, or a mix of maximization and minimization.

In multiobjective optimization, all the objectives must be optimized concurrently to get the real trade-off for decision-making. In this case, there is no single optimal solution, but rather a set of alternative solutions. These solutions are optimal in the wider sense that no other solutions in the search space are superior to them when all objectives are considered. They are known as Pareto-optimal solutions.

There are several conventional optimization-based algorithms for solving multiobjective optimization problems (Coello, 1999). These methods are not discussed here since none of them perform simultaneous optimization. Evolutionary algorithms (EAs) seem to be particularly suited for multiobjective optimization problems because they process a set of solutions in parallel, possibly exploiting similarities of solutions by recombination. Some researchers suggest that multiobjective search and optimization might be a problem area where EAs do better than other blind search strategies (Fonseca & Fleming, 1993; Valenzuela-Rendón, & Uresti-Charre, 1997). There are several EAs available in the literature those are capable of searching for multiple Pareto-optimal solutions concurrently in a single run. Some of the popular algorithms are: the Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985), Hajela and Lin's (1992) genetic algorithm (HLGA), Non-dominated Sorting Genetic Algorithms (NSGA) (Srinivas & Deb, 1994), Fonseca and Fleming's (1993) evolutionary algorithm (FFES), Niche Pareto Genetic Algorithm (NPGA) (Horn, Nafpliotis, & Goldberg, 1994), the Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler & Thiele, 1999), the Pareto Archived Evolution Strategy (PAES) (Knowles & Corne, 1999, 2000), and New Evolutionary Multiobjective Algorithms

(NEMA) (Sarker, Liang, & Newton, 2002). However, none of these algorithms performs consistently for all types of problems. Recently, we developed the *Pareto-based Differential Evolution* (PDE) approach, which outperforms most existing evolutionary multiobjective algorithms over continuous domains (Abbass & Sarker, 2002; Abbass, Sarker, & Newton, 2001; Sarker & Abbass, 2004).

Differential Evolution

Evolutionary algorithms (Fogel, 1995) are a kind of global optimization technique that uses selection and recombination as its primary operators to tackle optimization problems. Differential Evolution (DE) is a branch of evolutionary algorithms developed by Rainer Storn and Kenneth Price (Storn & Price, 1995) for optimization problems over continuous domains. In DE, each variable is represented in the chromosome by a real number. The approach works as follows:

1. Create an initial population of potential solutions at random, where repair rules guarantee that those variables' values are within their boundaries.
2. Until termination conditions are satisfied:
 - Select at random a trial individual for replacement, an individual as the main parent, and two individuals as supporting parents.
 - With some probability, each variable in the main parent is perturbed by adding to it a ratio, F , of the difference between the two values of this variable in the other two supporting parents. At least one variable must be changed. This process represents the crossover operator in DE.
 - If the resultant vector is better than the trial solution, it replaces it; otherwise the trial solution is retained in the population.
 - Go to 2 above.

From the previous discussion, DE differs from *genetic algorithms* (GA) in a number of points:

- DE uses real number representation while conventional GA uses binary, although GA sometimes uses integer or real number representation as well.
- In GA, two parents are selected for crossover and the child is a recombination of the parents. In DE, three parents are selected for crossover and the child is a perturbation of one of them.
- The new child in DE replaces a randomly selected vector from the population only if it is better than it. In conventional GA, children replace the parents with some probability regardless of their fitness.

A Differential Evolution Algorithm for MOPs

A generic version of the adopted algorithm can be found in Abbass and Sarker (2002). The PDE algorithm is similar to the DE algorithm with the following modifications:

1. The initial population is initialized according to a Gaussian distribution $N(0.5, 0.15)$.
2. The step-length parameter is generated from a Gaussian distribution $N(0, 1)$.
3. Reproduction is undertaken only among nondominated solutions in each generation.
4. The boundary constraints are preserved either by reversing the sign if the variable is less than 0 or subtracting 1 if it is greater than 1 until the variable is within its boundaries.
5. Offspring are placed into the population if they dominate the main parent.

The algorithm works as follows. An initial population is generated at random from a Gaussian distribution with mean 0.5 and standard deviation 0.15. All dominated solutions are removed from the population. The remaining non-dominated solutions are retained for reproduction. A child is generated from a selected three parents and is placed into the population if it dominates the first selected parent; otherwise a new selection process takes place. This process continues until the population is completed.

Proposed Algorithm

This section presents the nomenclatures and representations used in describing the algorithm and the details of the differential evolution algorithm for solving multiobjective optimization problems.

Nomenclatures

From herein, the following notations will be used for a single hidden layer MLP:

- I and H are the number of input and hidden units respectively.
- $\mathbf{X}^p \in \mathbf{X} = (x_1^p, x_2^p, \dots, x_I^p)$, $p = 1, \dots, P$, \mathbf{X}^p is the p^{th} pattern in the input feature space \mathbf{X} of dimension I , and P is the total number of patterns.
- Without any loss of generality, $\mathbf{Y}_o^p \in \mathbf{Y}_o$ is the corresponding scalar of pattern \mathbf{X}^p in the hypothesis space \mathbf{Y}_o .
- w_{ih} and w_{ho} , are the weights connecting input unit i , $i = 1, \dots, I$, to hidden unit h ,

$h = 1, \dots, H$, and hidden unit h to the output unit o (where o is assumed to be 1 in this research) respectively.

- $\Theta_h(\mathbf{X}^p) = \sigma(a_h)$; $a_h = \sum_{i=0}^I w_{ih} x_i^p$, $h = 1, \dots, H$, is the h^{th} hidden unit's output corresponding to the input pattern \mathbf{X}^p , where a_h is the activation of hidden unit h , and $\sigma(\cdot)$ is the activation function that is taken in this research to be the logistic function $\sigma(z) = \frac{1}{1 + e^{-Dz}}$, with D the function's sharpness or steepness and is taken to be 1 unless it is mentioned otherwise.
- $\hat{Y}_o^p = \sigma(a_o)$; $a_o = \sum_{h=0}^H w_{ho} \Theta(\mathbf{X}^p)$ is the network output and a_o is the activation of output unit o corresponding to the input pattern \mathbf{X}^p .

Representation

In deciding on an appropriate representation, we tried to choose a representation that can be used for other architectures without further modifications. Our chromosome is a class that contains one matrix Ω and one vector ρ . The matrix Ω is of dimension $(I + O) \times (H + O)$. Each element $\omega_{ij} \in \Omega$, is the weight connecting unit i with unit j , where $i = 0, \dots, (I-1)$ is the input unit i , $i = I, \dots, (I + O - 1)$ is the output unit $i-I$, $j = 0, \dots, (H-1)$ is the hidden unit j , and $j = H, \dots, (H + O - 1)$ is the output unit $(j - H)$. This representation has the following two characteristics that we are not using in the current version but can easily be incorporated in the algorithm for future work:

1. It allows direct connection from each input to each output units (we allow more than a single output unit in our representation).
2. It allows recurrent connections between the output units and themselves.

The vector ρ is of dimension H , where $\rho_h \in \rho$ is a binary value used to indicate if hidden unit h exists in the network or not; that is, it works as a switch to turn a hidden unit on or off. The sum, $\sum_{h=0}^H \rho_h$, represents the actual number of hidden units in a network, where H is the maximum number of hidden units. This representation allows both training the weights in the network as well as selecting a subset of hidden units.

Methods

We have a function-approximation problem that may arise in many situations including data mining, forecasting, and estimation. We have no prior knowledge about the nature of the function.

Based on the discussions in the first section, we have decided to use one input layer,

one hidden layer, and one output layer in the network. Our main objective is to estimate the connection weights by minimizing the total error and select the appropriate number of hidden nodes. In this chapter, we need to determine the connection weights that are real variables and select the hidden nodes in the network that are associated each with a binary variable (1 if the hidden unit exists and 0 for not). We set two objectives in this problem as follows:

1. Minimization of error
2. Minimization of number of hidden units in the ANN

The problem can be handled as a multiobjective optimization problem. The steps to solve this problem are given next.

1. Create a random initial population of potential solutions.
2. Until termination conditions are satisfied, repeat:
 - (a) Evaluate the individuals in the population and label those who are nondominated.
 - (b) Delete all dominated solutions from the population.
 - (c) Repeat:
 - Select at random an individual as the main parent and two individuals as supporting parents.
 - With some probability $Uniform(0,1)$, crossover the parents to generate a child where each weight in the main parent is perturbed by adding to it a ratio, $F \in Gaussian(0,1)$, of the difference between the two values of this variable in the two supporting parents. At least one variable must be changed.
 - If the child dominates the main parent, place it into the population.
 - (d) Until the maximum population size is reached.

Experiments

In this section, we provide the experimental setup, computation results, and discussions on results.

Experimental Setup

To test the performance of our proposed method, we experimented with a known polynomial function in two variables and of the third degree with noise. Noise was added to each input with a probability 0.2 from a Gaussian distribution $N(0,0.2)$. The function took the form

$$x_1^3 + x_2^3$$

We generated 2000 instances as a training set and another 2000 as a test set. Both training and test sets were generated independently. Variables were generated from a uniform distribution between 0 and 1. After computing the output of the function, noises were added to the inputs only.

For the evolutionary approach, an initial population size of 50 was used and the maximum number of objective evaluations was set to 20,000. The number of inputs and the maximum number of hidden nodes were chosen as 2 and 10 respectively. The PDE algorithm was run with five different crossover rates (0, 0.05, 0.1, 0.5, and 0.9) and five different mutation rates (0, 0.05, 0.1, 0.5, and 0.9). For each combination of crossover and mutation rates, results were collected and analyzed over 10 runs with different seed initializations. The initial population is initialized according to a Gaussian distribution $N(0,1)$. The step-length parameter F is generated for each variable from a Gaussian distribution $N(0,1)$. The algorithm is written in standard C++ and ran on a Sun Sparc 4.

The BP algorithm was tested for 10 different architectures created varying the hidden nodes from 1 to 10. For each architecture, the BP algorithm was run 10 times with 10 different random seeds. The same 10 seeds were used in all BP runs as well as the evolutionary approach.

Experimental Results and Discussions

For each architecture, the results from each of the 10 runs of the BP algorithm were recorded and analyzed. The performance of the evolutionary approach is measured by the average performance of the Pareto set, which is selected on the training set, on the test set. The average error rate from these 100 runs (10 architectures, each with 10 runs) is found to be 0.095 with a range of architecture-wise average 0.094 to 0.096. The average error rates with different crossover and mutation rates for the PDE approach are plotted in Figures 1 and 2. Each average error rate is a mean of 10 runs for a given crossover and mutation. In the x-axis of both figures, the numbers 1 to 5 represent the crossover or mutation rates from 0.0 to 0.9 as discussed earlier.

As seen in Figure 1, the error rate is minimal when the mutation rate is 0.1. At this mutation rate, the error rate varies within a narrow zone of 0.055 to 0.056. As shown in Figure 2, for all crossover rates, the error rate versus mutation rate follows a convex-like curve. Here the error rate decreases up to the minimum, where mutation rate is 0.10, and then increases. This nice pattern helps in choosing the optimum crossover and mutation rates.

Figure 1. Error rate vs. crossover rate

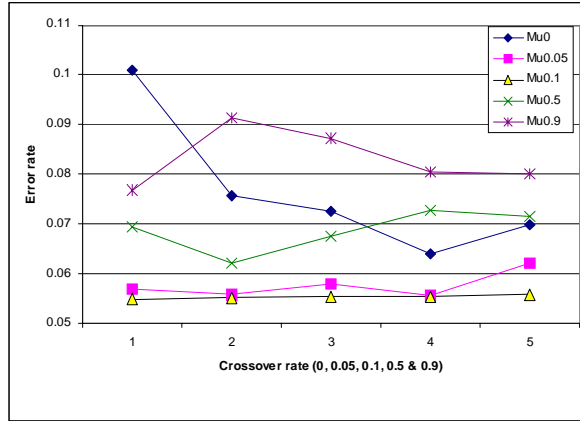
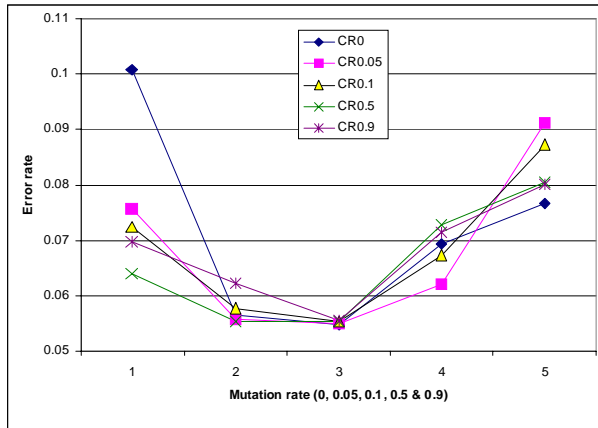


Figure 2. Error rate vs. mutation rate



In our case, a mutation rate of 0.10 and a crossover rate of 0.05 achieved the best performance with an error rate of 0.055. With this best crossover and mutation rates, the best error rate in a single run is 0.054 with three hidden nodes (a solution from the Pareto front).

Taking the average figures in both BP and PDE approaches, it is evident that the PDE approach reduces the error rate from 0.095 to 0.055, which is around 42% improvement. This emphasizes the advantages of the evolutionary approach in terms of accuracy and speed.

We need to emphasize here that the evolutionary approach performed in the same number of epochs better than what 10 different BP runs did. To explain this further, we needed to find the best number of hidden units on our test task. To do this, we trained 10 different neural networks with the number of hidden units ranging from 1 to 10. In the evolutionary

Table 1. Major ANN applications in finance and manufacturing

Finance	Manufacturing
<ul style="list-style-type: none"> • Stock performance and selection • Foreign exchange-rate forecasting • Corporate bankruptcy prediction • Fraud detection • Trading and forecasting • Future-price estimation 	<ul style="list-style-type: none"> • Condition monitoring • Tool wearing and breaking • Cost estimation • Fault diagnosis • Parameter selection • Quality control

approach, however, we set the maximum number of hidden units and the evolutionary approach determined the appropriate number without the need of experimenting with 10 different networks. In addition, the crossover is much faster than BP, adding more advantages to the evolutionary approach.

Applications in Finance and Manufacturing

A brief review of applications of traditional ANNs in finance and manufacturing is provided in chapter 1 of this book. The major application areas in finance (Kamruzzaman & Sarker, 2004a, 2004b) and manufacturing (Khan, Frayman, & Nahavandi, 2003, 2004) are provided in the following table.

Chapters 3 to 9 of this book present detailed applications of traditional ANNs to different finance case problems and chapters 10 to 15 provide applications for a number of different manufacturing operational problems. To the best of our knowledge, not only in this book but also in open literature, no finance and manufacturing applications used simultaneous evolution of network architectures and connection weights in ANNs. However, as the prediction performance of ANNs can be improved using the methodology presented in this chapter, we are certain that it would provide better results for finance and manufacturing applications.

Conclusion and Future Research

In this research, we investigated the simultaneous evolution of architectures and connection weights in ANNs. In so doing, we proposed the concept of multiobjective optimization to determine the best architecture and appropriate connection weights concurrently. The multiobjective optimization problem was then solved using the Pareto Differential Evolution algorithm. The result on a test problem was significantly better

when compared with BP. Although it shows a very promising performance, in our future work, we will need to experiment with more problems to generalize our findings. However, it is expected that it would provide better performances for most general cases.

Acknowledgment

This work is supported by the University of New South Wales at the Australian Defence Force Academy (UNSW@ADFA) Special Research Grants TERM6 2001 DOD02 ZCOM Z2844 awarded to Dr. H. Abbass and Dr. R. Sarker. The authors would like to thank the two editors of this book Dr. Joarder Kamruzzaman and Dr. Rezaul Begg for organizing a quick review of this chapter.

References

- Abbass, H., & Sarker, R. (2002). The Pareto differential evolution algorithm. *International Journal of Artificial Intelligence Tools*, 11(4), 531-552.
- Abbass, H. A., Sarker, R., & Newton, C. (2001). A Pareto differential evolution approach to vector optimisation problems. *IEEE Congress on Evolutionary Computation*, 2, 971-978.
- Basheer, I. (2000). Selection of methodology for modeling hysteresis behaviour of soils using neural networks. *Journal of Computer-Aided Civil and Infrastructure Engineering*, 5(6), 445-463.
- Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: Fundamentals, computing, design, and applications. *Journal of Microbiological Methods*, 43, 3-31.
- Coello, C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems: An International Journal*, 1, 269-308.
- Fahlman, S., & Lebiere, C. (1990). *The cascade correlation learning architecture* (CMU-CW-90-100). Pittsburgh, PA: Carnegie Mellon University.
- Fogel, D. B. (1995) *Evolutionary computation: Towards a new philosophy of machine intelligence*. New York: IEEE Press.
- Fonseca, C., & Fleming, P. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference of Genetic Algorithms* (pp. 416-423). San Mateo, CA: Morgan Kaufmann.
- Hajela, P., & Lin, C. Y. (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4, 99-107.
- Hecht-Nielsen, R. (1990). *Neurocomputing*. Reading, MA: Addison-Wesley.

- Hertz, J., Krogh, A., & Palmer, R. (1991). *Introduction to the theory of neural computation*. Reading, MA: Addison-Wesley.
- Horn, J., Nafpliotis, N., & Goldberg, D. (1994). A niched Pareto genetic algorithm for multiobjective optimisation. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, (pp. 82-87). Piscataway, NJ: IEEE Press.
- Kamruzzaman, J., & Sarker, R. (2004a). ANN based forecasting of foreign currency exchange rates. *Neural Information Processing — Letters and Reviews*, 3(2), 49-58.
- Kamruzzaman, J., & Sarker, R. (2004b). Application of support vector machine to forex monitoring. *IEEJ Transaction on Electronics, Information, and Systems*, 124-C(10), 1944-1951.
- Khan, M. I., Frayman, Y., & Nahavandi, S. (2003). Improving the quality of die casting by using artificial neural network for porosity defect modelling. In (A. K. Dahle (Ed.), *Proceedings of the 1st International Light Metals Technology Conference* (pp. 243-245). Brisbane, Australia: CAST Centre Pty Ltd.
- Khan, M. I., Frayman, Y., & Nahavandi, S. (2004). Knowledge extraction from a mixed transfer function artificial neural network. In R. Alo, V. Kreinovich, & M. Beheshti (Eds.), *Proceedings of the Fifth International Conference on Intelligent Technologies*. Houston, TX: University of Houston Downtown.
- Kim, K., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19, 125-132.
- Knowles, J., & Corne, D. (1999). The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. In *1999 IEEE Congress on Evolutionary Computation* (pp. 149-172). Washington, DC: IEEE Service Centre.
- Knowles, J., & Corne, D. (2000). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2), 149-172.
- LeCun, Y., Denker, J. J., & Solla, S. A. (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in neural information processing systems* (pp. 598-605). San Mateo, CA: Morgan Kaufmann.
- Sarker, R., & Abbass, H. (2004). Differential evolution for solving multi-objective optimization problems. *Asia-Pacific Journal of Operations Research*, 21(2), 225-240.
- Sarker, R., Liang, K., & Newton, C. (2002). A new evolutionary algorithm for multiobjective optimization. *European Journal of Operational Research*, 140(1), 12-23.
- Schaffer, J. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms* (pp. 93-100). Hillsdale, NJ: Lawrence Erlbaum.
- Srinivas, N., & Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3), 221-248.
- Storn, R., & Price, K. (1995). *Differential evolution: A simple and efficient adaptive*

scheme for global optimization over continuous spaces (TR-95-012). Berkeley, CA: International Computer Science Institute.

- Valenzuela-Rendón, M., & Uresti-Charre, M. (1997). A non-generational genetic algorithm for multiobjective optimization. In *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 658-665). San Mateo, CA: Morgan Kaufmann.
- Yao, X. (1999). Evolving artificial neural networks. In *Proceedings of the IEEE, Vol. 87, No. 9* (pp. 1423-1447).
- Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transaction on Neural Networks*, 8(3), 694-713.
- Yao, X., & Liu, Y. (1998). Making use of population information in evolutionary artificial neural networks. *IEEE Transaction on Systems, Man and Cybernetics*, 28(3), 417-425.
- Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transaction on Evolutionary Computation*, 3(4), 257-271.

SECTION II:
ANNs IN FINANCE

Chapter III

Neural Network-Based Stock Market Return Forecasting Using Data Mining for Variable Reduction

David Enke, University of Missouri – Rolla, USA

Abstract

Researchers have known for some time that nonlinearity exists in the financial markets and that neural networks can be used to forecast market returns. Unfortunately, many of these studies fail to consider alternative forecasting techniques, or the relevance of the input variables. The following research utilizes an information-gain technique from machine learning to evaluate the predictive relationships of numerous financial and economic input variables. Neural network models for level estimation and classification are then examined for their ability to provide an effective forecast of future values. A cross-validation technique is also employed to improve the generalization ability of the models. The results show that the classification models generate higher accuracy in forecasting ability than the buy-and-hold strategy, as well as those guided by the level-estimation-based forecasts of the neural network and benchmark linear regression models.

Introduction

Important changes have taken place over the last two decades within the financial markets, including the use of powerful communication and trading platforms that have increased the number of investors entering the markets (Elton & Gruber, 1991). Traditional capital market theory has also changed, and methods of financial analysis have improved (Poddig & Rehkugler, 1996). Stock-return forecasting has attracted the attention of researchers for many years and typically involves an assumption that fundamental information publicly available in the past has some predictive relationships to future stock returns or indices. The samples of such information include economic variables, exchange rates, industry- and sector-specific information, and individual corporate financial statements. This perspective is opposed to the general tenets of the efficient market hypothesis (Fama, 1970) which states that all available information affecting the current stock value is constituted by the market before the general public can make trades based on it (Jensen, 1978). Therefore, it is impossible to forecast future returns since they already reflect all information currently known about the stocks. This is still an empirical issue since there is contradictory evidence that markets are not fully efficient, and that it is possible to predict the future returns with results that are better than random (Lo & MacKinlay, 1988).

With this in mind, Balvers, Cosimano, and McDonald (1990), Breen, Glosten, and Jagannathan (1990), Campbell (1987), Fama and French (1988, 1989), Fama and Schwert (1977), Ferson (1989), Keim and Stambaugh (1986), and Schwert (1990), among others, provide evidence that stock market returns are predictable by means of publicly available information such as time-series data on financial and economic variables. These studies identify that variables such as interest rates, monetary-growth rates, changes in industrial production, and inflation rates are statistically important for predicting a portion of the stock returns. However, most of the studies just mentioned that attempt to capture the relationship between the available information and the stock returns rely on simple linear regression assumptions, even though there is no evidence that the relationship between the stock returns and the financial and economic variables is linear. Since there exists significant residual variance of the actual stock return from the prediction of the regression equation, it is possible that nonlinear models could be used to explain this residual variance and produce more reliable predictions of the stock price movements (Mills, 1990; Priestley, 1988).

Since many of the current modeling techniques are based on linear assumptions, a method of financial analysis that considers the nonlinear analysis of integrated financial markets needs to be considered. Although it is possible to perform a nonlinear regression, most of these techniques require that the nonlinear model must be specified before the estimation of parameters can be determined. Neural networks are a nonlinear modeling technique that may overcome these problems (Hill, O'Conner, & Remus, 1996). Neural networks offer a novel technique that does not require a prespecification during the modeling process since they independently learn the relationship inherent in the variables. This is especially useful in security investment and other financial areas where much is assumed and little is known about the nature of the processes determining asset prices (Burrell & Folarin, 1997). Neural networks also offer the flexibility of numerous

architecture types, learning algorithms, and validation procedures. Current studies that reflect recent interest in applying neural networks to answer future stock behaviors include Abhyankar, Copeland, and Wong (1997), Chenoweth and Obradovic (1996), Desai and Bharati (1998), Gencay (1998), Leung, Daouk, and Chen (2000), Motiwalla and Wahab (2000), Pantazopoulos, Tsoukalas, Bourbakis, Brun, and Houstis (1998), Qi and Maddala (1999), and Wood and Dasgupta (1996).

In addition to model-development issues, it has also been found that stock trading driven by a certain forecast with a small forecasting error may not be as profitable as trading guided by an accurate prediction of the sign of stock return (Aggarwal & Demaskey, 1997; Leung et al., 2000; Maberly, 1986; Wu & Zhang, 1997). Furthermore, given the existence of a vast number of articles addressing the predictabilities of stock market return, most of the proposed models rely on various assumptions and often employ a particular series of input variables without justification as to why they were chosen. A systematic approach to determine what inputs are important is necessary. Therefore, the following research will begin with a discussion of an information-gain data-mining technique for performing the variable-relevance analysis. Two neural network approaches that can be used for classification and level estimation will also be briefly reviewed, followed by a discussion of the neural network models, including the generalized regression, probabilistic, and multilayer feed-forward neural networks that were developed to estimate the value (level) and classify the direction (sign) of excess stock returns on the S&P 500 stock index portfolio. Five-fold cross validation and early-stopping techniques are also implemented in this study to improve the generalization ability of the feed-forward neural networks. The resulting data selection and model development, empirical results, and discussion and conclusion will then be presented. Data sources and descriptions are given in the Appendix.

Methodology for Data Selection

Whenever possible, large-scale deterministic components, such as trends and seasonal variations, should be eliminated from the inputs since the network will attempt to learn the trend and use it in the prediction (Nelson, Hill, Remus, & O'Conner, 1999; Pantazopoulos et al., 1998). Therefore, the data collected in this study, excluding *DIV*, *TI*, *SP*, *DY*, and *ER*, were seasonally adjusted. The source and definition of all the variables are given in the Appendix. In addition, due to the lag associated with the publication of macroeconomic indicators as mentioned by Qi and Maddala (1999), certain data, particularly *PP*, *IP*, *CP*, and *MI*, were included in the base set with a two-month time lag while the rest of the variables were included with a one-month time lag. This was done to simulate how this data would be received in the real setting, such that only observable, but not future, data would be provided as inputs to the forecasting models.

For this study, the differences [$P_t - P_{t-1}$] of the variables were provided to the networks so that different input variables can be compared in terms of relative change to the monthly stock returns, since the relative change of variables may be more meaningful to the models than the original values when forecasting a financial time series. Monthly data

from March 1976 to December 1999, for a total of 286 periods and for each of 31 financial and economic variables, were collected and analyzed. These variables, including PP_{t-1} , CP_{t-1} , IP_{t-1} , MI_{t-1} , $T3_t$, $T6_t$, $T12_t$, $T60_t$, $T120_t$, $CD1_t$, $CD3_t$, $CD6_t$, AAA_t , BAA_t , DIV_t , $T1_t$, SP_t , DY_t , $TE1_t$, $TE2_t$, $TE3_t$, $TE4_t$, $TE5_t$, $TE6_t$, $DE1_t$, $DE2_t$, $DE3_t$, $DE4_t$, $DE5_t$, $DE6_t$, and $DE7_t$, were primarily employed to predict the level and to classify the sign of the excess stock returns (ER_{t+1}) on the S&P 500 index portfolio. These data consisted of a mixture of the variables conducted by various researchers, including Desai and Bharati (1998), Leung et al. (2000), Motiwalla and Wahab (2000), and Qi and Maddala (1999). However, two variables often used in the literature, long-term treasury rates and commercial paper, were not applicable due to the fact that the 30-year treasury rate provided by the Federal Reserve Board of Governors started from February 1977, while the series of commercial papers had been discontinued because of a change in methodology in September 1997. Several financial instruments, such as CD and T-bill rates with additional maturities, were included to supplement unavailable data in this study.

While uncertainty in selecting the predictive variables to forecast stock returns still exists, as can be observed from a variety of input variables used in a recent literature survey, several techniques such as regression coefficients (Qi & Maddala, 1999), autocorrelations (Desai & Bharati, 1998), backward stepwise regression (Motiwalla & Wahab, 2000), and genetic algorithms (Motiwalla & Wahab, 2000) have been employed by a few studies to perform variable subset selection. In addition, several researchers, such as Leung et al. (2000), Gencay (1998), and Pantazopoulos et al. (1998), have subjectively selected the subsets of variables based on empirical evaluations. None of these studies have incorporated all available variables previously mentioned in the literature to uncover the predictive input variables, while at the same time eliminating irrelevant or redundant data. It is critical to consider all the data since leaving out relevant variables or keeping irrelevant variables may be detrimental, causing confusion to the neural network models. Besides, the use of too many variables would require a neural network that contains unnecessary neurons and hidden layers. Unfortunately, there is no consistent method that has been used to pick out the useful variables in stock return forecasting. This may be due to the fact that the behavior of this data is not well known.

One alternative that can be used to extract valuable information and knowledge from large amounts of data involves the use of data mining (Han & Micheline, 2000). Specifically, there have been studies in the various areas of data mining (i.e., machine learning, fuzzy logic, statistics, and rough-set theories) on variable relevance analysis. Relevance analysis can also be performed on financial data with the aim of removing any irrelevant or redundant variables from the learning process. The general idea behind variable relevance analysis is to compute some measures that can be used to quantify the relevance of variables hidden in a large data set with respect to a given class or concept description. Such measures include information gain, the Gini index, uncertainty, and correlation coefficients. For this research, an inductive learning decision tree algorithm that integrates an information gain analysis technique with a dimension-based data analysis method was selected as it can be effectively used for variable subset selection (Han & Micheline, 2000). The resulting method removes the less information producing variables and collects the variables that contain more information. Therefore, it may be the most appropriate data-mining technique to perform variable subset selection when the usefulness of the data is unknown. While using the information gain analysis

technique, the predicted directions of excess stock returns were used as class distributions. The resulting variables with the high information gain were chosen as the relevance input variables provided to the neural network models. The following paragraphs give an introduction to the information-gain calculation. It is recommended that readers who are interested in full details of the information gain algorithm should refer to Quinlan (1993).

Let S be a set consisting of s data samples. Suppose the class label variable has m distinct values defining m distinct classes, C_i (for $i = 1, 2, \dots, m$). Let s_i be the number of samples of S in class C_i . The expected information for classification is given by:

$$I(s_1, s_2, s_3, \dots, s_m) = - \sum_{i=1}^m p_i \log_2(p_i) \quad (1)$$

where p_i is the probability that an arbitrary sample belongs to class C_i and is estimated by s_i/s . Note that a log function to the base 2 is used since the information is encoded in bits. Let variable A have v distinct values denoted in order from small to large values as $\{a_1, a_2, a_3, \dots, a_v\}$. Any split value lying between a_i and a_{i+1} will have the same effect of dividing the samples into those whose value of the variable A lies in $\{a_1, a_2, a_3, \dots, a_i\}$ and those whose value is in $\{a_{i+1}, a_{i+2}, a_{i+3}, \dots, a_v\}$. However, the midpoint of each interval is usually chosen as the representative split. It is defined as $(a_i + a_{i+1})/2$. Thus, there are $v-1$ possible splits on A , all of which are examined. Note that examining all $v-1$ splits is necessary to determine the highest information gain of A .

Variable A can therefore be used to partition S into 2 subsets, $\{S_j, S_2\}$, where S_j contains those samples in S that have values $\{a_1, a_2, a_3, \dots, a_i\}$ or $\{a_{i+1}, a_{i+2}, a_{i+3}, \dots, a_v\}$ of A . Let S_j contain s_{ij} samples of class C_i . The expected information based on this partitioning by A , also known as the "entropy" of A , is given by:

$$E(A) = \sum_{j=1}^v \frac{s_{1j} + s_{2j} + \dots + s_{mj}}{s} I(s_{1j}, s_{2j}, \dots, s_{mj}). \quad (2)$$

The term $(s_{1j} + s_{2j} + \dots + s_{mj})/s$ acts as the weight of the j^{th} subset and is the number of samples in the subset (i.e., having value a_j of A) divided by the total number of samples in S . Note that for a given subset S_j ,

$$I(s_{1j}, s_{2j}, \dots, s_{mj}) = - \sum_{i=1}^m p_{ij} \log_2(p_{ij}) \quad (3)$$

where $p_{ij} = s_{ij}/|S_j|$ and is the probability that a sample in S_j belongs to class C_i . The information gain obtained by this partitioning of the split on A is defined by:

$$Gain(A) = I(s_1, s_2, s_3, \dots, s_m) - E(A). \quad (4)$$

In this approach to relevance analysis, the highest information gain for each of the variables defining the samples in S can be obtained. The variable with the highest information gain is considered the most discriminating variable of the given set. By computing the information gain for each variable, a ranking of the variables can be obtained. Finally, the relevance threshold was determined to select only the strong relevance variables to be used in the forecasting models, and was chosen to eliminate the variables that contributed less than 0.1% of the total variation in the data set. This number relates to previous research with principle component analysis, and is also a result of trial-and-error. It allows the network to train efficiently, and also cuts the input data set in half.

For this research, each of the neural network models was compared against a linear regression model, as well as a buy-and-hold strategy. For all models, the data set used in this study was divided into two periods: The first period runs from March 1976 to Oct 1992 for a total of 200 months while the second period runs from November 1992 to December 1999 for a total of 86 months. The former was used for determining the specifications of the models and parameters of the forecasting techniques. The latter was reserved for out-of-sample evaluation and comparison of performances among the forecasting models.

Neural Network Models

Neural networks mimic the human brain and are characterized by the pattern of connections between the various network layers, the numbers of neurons in each layer, the learning algorithm, and the neuron activation functions. Generally speaking, a neural network is a set of connected input and output units where each connection has a weight associated with it. During the learning phase, the network learns by adjusting the weights so as to be able to correctly predict or classify the output target of a given set of input samples. Given the numerous types of neural network architectures that have been developed in the literature, three important types of neural networks were implemented in this study to compare their predictive ability against the classical linear regression model. The following three subsections give a brief introduction of these three neural network models.

Multilayer Feed-Forward Neural Network

Multilayer feed-forward neural networks have been widely used for financial forecasting due to their ability to correctly classify and predict the dependent variable (Vellido, Lisboa, & Vaughan, 1999). Backpropagation is by far the most popular neural network training algorithm that has been used to perform learning for multilayer feed-forward

neural networks. Since the feed-forward neural networks are well known and described elsewhere, the network structures and backpropagation algorithms are not described here. However, readers who are interested in greater detail can refer to earlier chapters or to Rumelhart and McClelland (1986) for a comprehensive explanation of the backpropagation algorithm used to train multilayer feed-forward neural networks.

During neural network modeling, Malliaris and Salchenberger (1993) suggest that validation techniques are required to identify the proper number of hidden layer nodes, thus avoiding underfitting (too few neurons) and overfitting (too many neurons) problems. Generally, too many neurons in the hidden layers results in excessive connections, resulting in a neural network that memorizes the data and lacks the ability to generalize. One approach that can be used to avoid over-fitting is n -fold cross-validation (Peterson, St Clair, Aylward, & Bond, 1995). A five-fold cross-validation, which was used in this experiment, can be described as follows: The data sample is randomly partitioned into five equal-sized folds and the network is trained five times. In each of the training passes, one fold is omitted from the training data and the resulting model is validated on the cases in that omitted fold, which is also known as a validation set. The first period (200 months) of the data set is used for the five-fold cross-validation experiment, leaving the second period for truly untouched out-of-sample data. The average root-mean-squared error over the five unseen validation sets is normally a good predictor of the error rate of a model built from all the data.

Another approach that can be used to achieve better generalization in trained neural networks is called early stopping (Demuth & Beale, 1998). This technique can be effectively used with the cross-validation experiment. The validation set is used to decide when to stop training. When the network begins to over-fit the data, the error on the validation cases will typically begin to rise. In this study, the training was stopped when the validation error increased for five iterations, causing a return of the weights and biases to the minimum of the validation error. The average error results of the validation cases (40 months in each fold for this study) from the n -fold cross-validation experiment are then used as criteria for determining the network structure, namely the number of hidden layers, number of neurons, learning algorithms, learning rates, and activation functions.

Generalized Regression Neural Network

While a number of articles address the ability of multilayer feed-forward neural network models for financial forecasting, none of these studies has practically applied the generalized regression neural network (GRNN) to forecast stock returns. Similar to the feed-forward neural networks, the GRNN can be used for function approximation to estimate the values of continuous dependent variables, such as future position, future values, and multivariable interpolation. The GRNN is a kind of radial-basis-function network and also looks similar to a feed-forward neural network responding to an input pattern by processing the input variables from one layer to the next with no feedback paths (Specht, 1991). However, its operation is fundamentally different. The GRNN is based on nonlinear regression theory that can be used when an assumption of linearity is not justified.

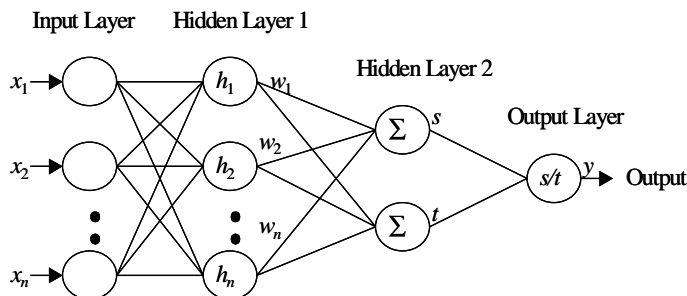
The training set contains the values of x (independent variables) that correspond to the value of y (dependent variable). This regression method will produce the optimal expected value of y , which minimizes the mean-squared error. The GRNN approach uses a method that frees the necessity to assume a specific functional form, allowing the appropriate form to be expressed as a probability density function that is empirically determined from observed data using the window estimation (Parzen, 1962). Therefore, this approach is not limited to any particular forms and requires no prior knowledge of the estimated function. The GRNN formula is briefly described as follows:

$$E[y/x] = \frac{\int_{-\infty}^{\infty} yf(x, y)dy}{\int_{-\infty}^{\infty} f(x, y)dy} \tag{5}$$

where y is the output of the estimator, x is the estimator input vector, $E[y/x]$ is the expected value of y given x , and $f(x, y)$ is the known joint continuous probability density function of x and y . When the density $f(x, y)$ is not known, it will be estimated from a sample of observations of x and y . For a nonparametric estimate of $f(x, y)$, the class of consistent estimators proposed by Parzen (1962) is used. As a result, the following equation gives the optimal expected value of y :

$$y = \frac{\sum_{i=1}^n h_i w_i}{\sum_{i=1}^n h_i} \tag{6}$$

Figure 1. Generalized regression neural network architecture



where w_i is the target output corresponding to the input training vector x_i and the output y , $h_i = \exp[-D_i^2 / (2\sigma^2)]$ is the output of hidden neuron, $D_i^2 = (x-u_i)^T(x-u_i)$ is the squared distance between the input vector x and the training vector u , and σ is a smoothing parameter of the radial basis function. The GRNN architecture is shown in Figure 1. The neuron of the hidden layer 1 is created to hold the input vector. The weight between the newly created hidden neuron and the neuron of the hidden layer 2 is assigned the target value.

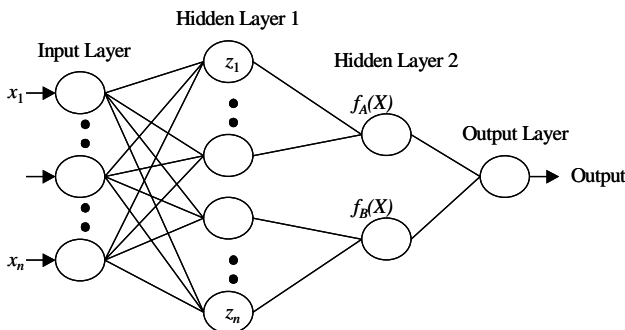
Probabilistic Neural Network

In contrast to the GRNN used to estimate the values of continuous variables, the probabilistic neural network (PNN) finds decision boundaries between categories of patterns. Therefore, the PNN is mainly used for classification problems and has been successfully used for classifying the direction of financial time series (Thawornwong, Enke, & Dagli, 2001). The PNN is a parallel implementation of a standard Bayesian classifier and has a four-layer network that can perform pattern classification. It is based essentially on the estimation of probability density functions for various classes as learned from training samples. The PNN learns from the sample data instantaneously and uses these probability density functions to compute the nonlinear decision boundaries between classes in a way that approaches the Bayes optimal (Specht, 1990). The PNN formula is explained as follows:

$$f_A(x) = \frac{1}{(2\pi)^{p/2} \sigma^p n} \sum_{i=1}^n z_i \tag{7}$$

where $f_A(x)$ is the probability density function estimator for class A, p is the dimensionality of training vector, $z_i = \exp[-D_i / (2\sigma^2)]$ is the output of hidden neuron, $D_i = (x - u_i)^T(x - u_i)$ is the distance between the input vector x and the training vector u from category A, and σ is a smoothing parameter.

Figure 2. Probabilistic neural network architecture



Theoretically, the PNN can classify an out-of-sample data with the maximum probability of success when enough training data is given (Wasserman, 1993). Figure 2 presents the PNN architecture. When an input is presented to the hidden layer 1, it computes distances from the input vector to the training vectors and produces a vector whose elements indicate how close the input is to the vectors of the training set. The hidden layer 2 then sums these elements for each class of inputs to produce a vector of probabilities as its net output. Finally, the activation function of the PNN output layer picks the maximum of these probabilities and classifies it into specific output classes.

Data Selection and Model Development

The selection of the input variables is a modeling decision that can greatly affect the model performance. For the neural network modeling, an information-gain data-mining analysis was used to find good subsets of the full set of the first-period input variables. Of the 31 variables, 15 variables were selected by the information gain data mining analysis as strong relevance predictors for the data set used in this study. They include *M1, T3, T6, T120, CD1, CD3, CD6, SP, TE2, TE3, TE4, DE2, DE3, DE5, and DE7*. Thus, these variables were consistently used as the input variables for training the neural networks throughout the modeling stage. The values of the input variables were first preprocessed by normalizing them within a range of -1 and $+1$ to minimize the effect of magnitude among the inputs, thereby increase the effectiveness of the learning algorithm.

It is well known that most trading practices adopted by financial analysts rely on accurate prediction of the price levels of financial instruments. Nonetheless, some recent studies have suggested that trading strategies guided by forecasts on the direction of the change in price level may be more effective and thus can generate higher profits. Aggarwal and Demaskey (1997) report that the performance of cross hedging improves significantly if the direction of changes in exchange rates can be predicted. In another study, Maberly (1986) explores the relationship between the direction of interday and intraday price changes on the S&P 500 futures. Wu and Zhang (1997) investigate the predictability of the direction of change in the future spot exchange rate. Leung et al. (2000) found that the forecasting models based on the direction of stock return outperform the models based on the level of stock return in terms of predicting the direction of stock market return and maximizing profits from investment trading.

The previously cited studies demonstrate the usefulness of forecasting the direction of change in the price or return level by means of a gain or a loss. In fact, the results of these findings are reasonable because accurate price estimation, as determined by its deviation from the actual observation, may not be a good predictor of the direction of change in the price levels of a financial instrument. To facilitate a more effective forecast, the two forecasting approaches, namely classification and level estimation, were investigated to evaluate the resulting performances of the model development. Specifically, the feed-forward neural networks were developed to both estimate the value (level) and classify the direction (sign) of excess stock returns on the S&P 500 index portfolio. For this study,

the GRNN was used to estimate the level of excess stock return, while the PNN was employed to classify the sign of excess stock return. Finally, the conventional linear regression model was developed to serve as a benchmark for performance comparison purposes. Note that the second period test data were never used during the model development so that these forecasting models were always tested on truly untouched out-of-sample data.

Neural Network Models for Level Estimation

For the feed-forward neural network using the backpropagation learning algorithm, a sigmoid hyperbolic tangent function was selected as the activation function to generate an even distribution over the input values. A single hidden layer was chosen for the neural network model since it has been successfully used for financial classification and prediction (*Swales & Yoon, 1992*). Accordingly, the feed-forward neural network was built with three layers (input layer, hidden layer, and output layer). Each of the relevant 15 input variables was assigned a separate input neuron to the input layer of the feed-forward neural network. One output neuron was used in the output layer to represent the predicted excess stock return of a given set of the 15 input variables. In this study, the connection weights were initially randomized and then determined during the backpropagation training process.

After numerous experiments with various numbers of hidden-layer neurons, learning algorithms, and learning rates, the feed-forward neural network employing 15 neurons in the input layer, 21 neurons in the hidden layer, 0.2 learning rate, and a resilient backpropagation learning algorithm (*Riedmiller & Braun, 1993*) was found to be the best network architecture based on the lowest average root-mean-squared error (RMSE) over the five-fold cross-validation experiment. In other words, this network architecture generated the lowest average RMSE over the five omitted folds (validation sets) in this study. The RMSE used in the feed-forward neural network for level estimation is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - t_i)^2} \quad (8)$$

where y_i is the predicted excess stock return, t_i is the actual excess stock return, and n is the number of validation cases (40 in this study). The average RMSE results were calculated only after the neural network outputs have been scaled back to their normal values. By conducting the five-fold cross-validation experiment, the forecasting results will not be based on a single network output because five neural network models were developed from the five different data sets. For this reason, the predicted excess stock returns of the five network outputs were averaged to generate the weighted excess return in this experiment.

Table 1. Portfolio neural network model for level estimation

Omitted Folds	Input-layer Neurons	Hidden-layer Neurons	Learning Rate
1	15	23	0.3
2	15	27	0.2
3	15	24	0.3
4	15	11	0.2
5	15	21	0.2

To further improve the forecasting performance, we also examined a portfolio network model consisting of the network architecture that produced the lowest RMSE in each omitted fold cross-validation experiment. In other words, the neural network model generating the lowest RMSE from each omitted fold experiment was chosen as one of the five neural networks deliberately combined as the portfolio network model. The resulting portfolio network architectures using the lowest RMSE in each omitted fold experiment are provided in Table 1. It is observed that the suitable neurons used in the hidden layer of the five combined portfolio networks that were trained based on different omitted folds are different. This observation suggests the importance of network modeling for a separate omitted fold experiment because the potentially better trained neural network may be obtained from the specific validation cases. Again, the *WER* of the portfolio network model was calculated from the five combined portfolio network outputs.

Unlike the feed-forward neural networks, the GRNN can be designed very quickly, and no early stopping technique is required during its training. Therefore, there would be no need to randomly partition the data into equal-sized folds for cross-validation. This allowed the first period (200 months) of the data set to be used in network training for predicting the excess stock returns of the last 86 months. In this study, a smoothing parameter of the radial-basis function equal to 1.00 was selected to approximate the network function more efficiently. The GRNN training process employed the same input variables, preprocessing techniques, and postprocessing techniques as those of the feed-forward neural network models.

Neural Network Models for Classification

Other than the output-layer structure, the feed-forward neural network for classification employed the same network structures as those used for level estimation. Since there are two classes for the sign of excess stock return, two output neurons were employed for the output layer to represent the different classes of the predicted excess stock return. For this research, the $[+1 -1]$ and $[-1 +1]$ classes represented the predicted positive and negative signs of excess stock return, respectively. The output neuron with the highest value was taken to represent the predicted sign of excess stock return based on a given set of the 15 input variables.

During testing, a feed-forward neural network employing 15 neurons in the input layer, 27 neurons in the hidden layer, 0.3 learning rate, and a resilient backpropagation learning

Table 2. Portfolio neural network model for classification

Omitted Folds	Input-layer Neurons	Hidden-layer Neurons	Learning Rate
1	15	21	0.3
2	15	19	0.2
3	15	28	0.3
4	15	27	0.3
5	15	23	0.3

algorithm was found to be the best network architecture with the lowest average RMSE over the five-fold cross-validation experiment. The RMSE used in the feed-forward neural network for classification is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{2n} \sum_{i=1}^n \{(y_{1i} - t_{1i})^2 + (y_{2i} - t_{2i})^2\}} \quad (9)$$

where y_1 and y_2 are the predicted classes of excess stock return of the two output neurons, t_1 and t_2 are the actual classes of excess stock return, and n is the number of validation cases. Just like the previously developed feed-forward neural network models for level estimation, the forecasting results will be based on five network outputs. Therefore, the majority of the signs of five network outputs are used to determine the decisive predicted sign of excess stock return. For example, when the five network models generate three positive predicted signs and two negative predicted signs of excess stock return based on a given set of the 15 input variables, the decisive predicted sign of excess stock return is resolved to be positive.

In addition, a portfolio network model for classification that consists of the network architecture producing the lowest RMSE in each omitted fold cross-validation experiment was explored. The resulting portfolio network architectures using the lowest RMSE in each omitted fold experiment are given in Table 2. As can be seen, the suitable hidden layer neurons of the five combined portfolio networks are different, implying a similar observation to those of the portfolio network model for level estimation. Similarly, the decisive predicted sign of excess stock return of the portfolio network model was derived from the majority of the five combined portfolio network outputs.

Like the GRNN, the design of the PNN is fast and straightforward. In fact, neither training nor an early stopping technique is required during its design. Therefore, the first period (200 months) of the data set was used in the network modeling for predicting the sign of the excess stock returns of the last 86 months. Also, a smoothing parameter equal to 1.00 was selected to consider several nearby design vectors. Again, the PNN design employed the same input variables and preprocessing techniques as those of the feed-forward neural network models.

Linear Regression for Level Estimation

For the linear regression forecasting, the backward stepwise regression for dimensionality reduction was employed to assume a linear additive relationship. This method started with the full set of variables in the model. The worst of the original variables was determined and removed from the full set. At each subsequent iteration or step, the worst of the remaining variables was removed from the last updated set. The significant t -statistics were used as criteria for retention of the significant input variables in the linear regression model. The remaining variables were thus used in predicting excess stock returns. In this study, the backward stepwise technique kept 10 variables, PP , MI , $T3$, $T12$, $T60$, $CD1$, $CD6$, BAA , SP , and $DE7$, as the significant input variables in the regression model ($\alpha = 0.05$). The regression model has the following function:

$$ER_{t+1} = -0.444 + (0.959 \times PP_{t-1}) + (0.100 \times MI_{t-1}) + (2.525 \times T3_t) + (5.981 \times T12_t) + (-4.584 \times T60_t) + (-1.050 \times CD1_t) + (-5.472 \times CD6_t) + (-1.437 \times BAA_t) + (-0.027 \times SP_t) + (8.295 \times DE7_t) \quad (10)$$

where all the regression coefficients are significant and the F -statistic is 2.027 (p -value 0.033), indicating that these forecasting variables contain information about future excess stock returns (F -critical = 1.91). The regression model shows that the relative changes of PP , MI , $T3$, $T12$, and $DE7$ have a positive effect on predictions of excess stock return, whereas the effect on excess stock returns of $T60$, $CD1$, $CD6$, BAA , and SP is negative.

Results

The predictive performances of the developed models were evaluated using the untouched out-of-sample data (second period). This is due to the fact that the superior in-sample performance does not always guarantee the validity of the forecasting accuracy. One possible approach for evaluating the forecasting performance is to investigate whether traditional error measures such as those based on the RMSE or correlation (CORR) between the actual out-of-sample returns and their predicted values are small or

Table 3. Testing set performance measures

		CORR	RMSE	SIGN
Level Estimation Models	Original Level NN	0.0231	1.1614	0.6628*
	Portfolio Level NN	0.0528	1.1206	0.6860*
	GRNN	0.0714	1.1206	0.6860*
	Regression	0.0300	1.4467	0.4767
Classification Models	Original Class NN	0.2300	1.2200	0.6279*
	Portfolio Class NN	0.3150	1.0997	0.6977*
	PNN	0.3020	1.2575	0.6047*

highly correlate, respectively. However, there is some evidence in the literature suggesting that traditional measures of forecasting performance may not be strongly related to profits from trading (Pesaran & Timmermann, 1995). An alternative approach is to look at the proportion of time that the signs of excess stock returns (SIGN) are correctly predicted. In fact, Leitch and Tanner (1991) state that the forecast performance based on the sign measure matches more closely to the profit performance than do traditional criteria.

Table 3 reports all the three performance measures of the original level estimation feed-forward neural network (NN) using the lowest average RMSE (Original level NN), the portfolio level estimation feed-forward neural network using the lowest RMSE in each omitted fold (Portfolio level NN), the GRNN, the linear regression model (Regression), the original classification feed-forward neural network using the lowest average RMSE (Original Class NN), the portfolio classification feed-forward neural network using the lowest RMSE in each omitted fold (Portfolio Class NN), and the PNN from November 1992 to December 1999. RMSE in Table 3 represents the root-mean-squared error between the actual and predicted signs of excess stock return. CORR refers to the Pearson correlation coefficient between the actual and predicted signs of excess stock return (Pesaran & Timmermann, 1992). SIGN denotes the proportion of times the predicted signs of excess stock returns are correctly classified. Note that the +1 and -1, representing the positive and negative decisive predicted signs from the PNN and the classification feed-forward neural networks, were used to compute the resulting classification performances in the study. To compare the classification performances with those of the Regression, the GRNN, and the feed-forward neural networks for level estimation, the original RMSE and CORR performance measures of these level estimation models were recalculated in connection with the signs of +1 and -1 of the classification models. That is, when the level-estimation models generate a positive predicted value of excess stock return, it will be converted to +1, or vice versa. The reason for this recalculation is that the PNN model is designed to give the exact signs of +1 and -1. Therefore, the prediction of the other forecasting models is required to adjust for unbiased performance comparisons.

According to Table 3, the empirical results show that neither the classification nor the level-estimation neural network models can accurately predict the signs of excess stock return because of the relatively low correlation relationship, although each of these models, except the Original Level NN model, is unquestionably better than the model using linear regression. This is due to the fact that the CORR of these models indicates higher positive relationship between the actual and predicted signs of excess stock return. It is also observed that the CORR of the classification models is constantly better than that of the level estimation models. In particular, the Portfolio Class NN has the highest CORR (0.3150) that can be obtained from the experiment. This reveals that the neural networks, especially the classification models, perform more accurately in correctly predicting the portion of future excess stock returns.

Regarding the second performance measure, the results again confirm that the linear regression model is the least accurate performer because it generates the highest RMSE (1.4467) compared to that of the neural network models. In contrast, the Portfolio Class NN model produces the lowest RMSE (1.0997). Nonetheless, the remaining two classification models, the Original Class NN and PNN models, signal slightly higher RMSE

results than those of the level-estimation neural network models. For the third performance measure, the results show that the percentage of the correct signs (SIGN) generated by the neural network models is far more accurate and consistently predictive than that of the linear regression forecast. This is because the correct signs produced by all of the neural network models are always greater than 0.6047. For statistical evaluation, the null hypothesis of no predictive effectiveness was calculated by conducting a one-sided test of $H_o: p = 0.50$ against $H_a: p > 0.50$. The SIGN marked with an asterisk (*) in Table 3 indicates the significant differences from the benchmark of 0.5 at a 95% level of confidence. More importantly, the Portfolio Class NN model once again signals the highest SIGN (0.6977) obtainable from the study, whereas the linear regression forecast has obtained only 0.4767 of the correct signs. This result verifies that the correct signs generated by each neural network model are better than random. In summary, the overall out-of-sample forecasts using the GRNN and Portfolio Class NN models are more accurate than those using the Original Level NN and Portfolio Level NN, Original Class NN, PNN, and Regression models with respect to their approaches. Particularly, the Portfolio Class NN model is proven to be the best performer in all of the performance measures used in this study. These findings strongly support the nonlinearity relationship between the past financial and economic variables and the future stock returns in the financial markets.

Discussion and Conclusion

An attempt has been made in this study to investigate the predictive power of financial and economic variables by adopting the variable-relevance-analysis technique in machine learning for data mining. This approach seems particularly attractive in selecting the variables when the usefulness of the data is unknown, especially when nonlinearity exists in the financial market as found in this study. Since it is known that the determinant between the variables and their interrelationships over stock returns could change over time, different relevance input variables may be obtained by conducting this data-mining technique under different time periods. In particular, we examined the effectiveness of the neural network models used for level estimation and classification, and noted the differences.

The feed-forward neural network training is usually not very stable since the training process may depend on the choice of a random start. Training is also computationally expensive in terms of the training times used to figure out the appropriate network structure. The degree of success, therefore, may fluctuate from one training pass to another. The empirical findings in this study show that our proposed development of the portfolio network models using the n -fold cross-validation and early stopping techniques does not sacrifice any of the first-period data used for training and validating the networks. This is especially useful when the data size is limited. In particular, we find that the method for improving the generalization ability of the feed-forward neural networks, a combination of n -fold cross-validation and early stopping techniques, clearly help improve the out-of-sample forecasts. In addition to the early stopping advantage,

improvement may be due to the fact that five-time network modeling allows the networks to extract more useful information from the data. Thus, the prediction based on the weighted excess return or the majority of excess return sign could effectively be used to reduce the prediction error. As a result, the portfolio network models for both classification and level estimation consistently outperform the linear regression, the generalized regression neural network, the probabilistic neural network, and the buy-and-hold account.

In conclusion, both researchers and practitioners have studied stock market prediction for many years. Many studies conclude that stock returns can be predicted by some financial and economic variables. To this end, our finding suggests that financial forecasting is always and will remain difficult since such data are greatly influenced by economical, political, international, and even natural events. Obviously, this study covers only fundamental available information, while the technical analysis approach remains intact. It is far from perfect as the technical analysis has been proved to provide invaluable information during stock-price and stock-return forecasting, and to some extent has been known to offer a relative mixture of human, political, and economical events. In fact, there are many studies done by both academics and practitioners in this area. If both technical and fundamental approaches are thoroughly examined and included during the variable relevance analysis modeling, it would no doubt be a major improvement in predicting stock returns. This study did not consider profitability and assumes that any trading strategies of investing in either the stock index portfolio or risk-free account will occur in the absence of trading costs. Future research should consider profitability and trading simulation under the scenarios of stock dividends, transaction costs, and individual tax brackets to replicate the realistic investment practices.

Acknowledgments

The author would like to acknowledge the contributions of Suraphan Thawornwong. Dr. Thawornwong was instrumental with model development, testing, and the preparation of this chapter.

References

- Abhyankar A., Copeland, L. S., & Wong, W. (1997). Uncovering nonlinear structure in real-time stock-market indexes: The S&P 500, the DAX, the Nikkei 225, and the FTSE-100. *Journal of Business & Economic Statistics*, 15, 1-14.
- Aggarwal, R., & Demaskey, A. (1997). Using derivatives in major currencies for cross-hedging currency risks in Asian emerging markets. *Journal of Future Markets*, 17, 781-796.

- Balvers, R. J., Cosimano, T. F., & McDonald, B. (1990). Predicting stock returns in an efficient market. *Journal of Finance*, 55, 1109-1128.
- Breen, W., Glosten, L. R., & Jagannathan, R. (1990). Predictable variations in stock index returns. *Journal of Finance*, 44, 1177-1189.
- Burrell, P. R., & Folarin, B. O. (1997). The impact of neural networks in finance. *Neural Computing & Applications*, 6, 193-200.
- Campbell, J. (1987). Stock returns and the term structure. *Journal of Financial Economics*, 18, 373-399.
- Chenoweth, T., & Obradovic, Z. (1996). A multi-component nonlinear prediction system for the S&P 500 Index. *Neurocomputing*, 10, 275-290.
- Demuth, H., & Beale, M. (1998). *Neural Network Toolbox: For use with MATLAB* (5th ed.). Natick, MA: The Math Works, Inc.
- Desai, V. S., & Bharati, R. (1998). The efficiency of neural networks in predicting returns on stock and bond indices. *Decision Sciences*, 29, 405-425.
- Elton, E. J., & Gruber, M. J. (1991). *Modern Portfolio Theory and Investment Analysis* (4th ed.). New York: John Wiley & Sons.
- Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, 25, 383-417.
- Fama, E. F., & French, K. R. (1988). Dividend yields and expected stock returns. *Journal of Financial Economics*, 22, 3-25.
- Fama, E. F., & French, K. R. (1989). Business conditions and expected returns on stocks and bonds. *Journal of Financial Economics*, 25, 23-49.
- Fama, E. F., & Schwert, W. G. (1977). Asset returns and inflation. *Journal of Financial Economics*, 5, 115-146.
- Ferson, W. (1989). Changes in expected security returns, risk, and the level of interest rates. *Journal of Finance*, 44, 1191-1217.
- Gencay, R. (1998). Optimization of technical trading strategies and the profitability in securities markets. *Economics Letters*, 59, 249-254.
- Han, J., & Micheline, K. (2000). *Data mining: Concepts and techniques*. San Francisco: Morgan Kaufmann.
- Hill, T., O'Connor, M., & Remus, W. (1996). Neural network models for time series forecast. *Management Science*, 42, 1082-1092.
- Jensen, M. (1978). Some anomalous evidence regarding market efficiency. *Journal of Financial Economics*, 6, 95-101.
- Keim, D., & Stambaugh, R. (1986). Predicting returns in the stock and bond markets. *Journal of Financial Economics*, 17, 357-390.
- Leitch, G., & Tanner, J. E. (1991). Economic forecast evaluation: Profits versus the conventional error measures. *American Economic Review*, 81, 580-590.
- Leung, M. T., Daouk, H., & Chen, A. S. (2000). Forecasting stock indices: A comparison of classification and level estimation models. *International Journal of Forecasting*, 16, 173-190.

- Lo, A. W., & MacKinlay, A. C. (1988). Stock market prices do not follow random walks: Evidence from a simple specification test. *Review of Financial Studies*, 1, 41-66.
- Maberly, E. D. (1986). The informational content of the interday price change with respect to stock index futures. *Journal of Futures Markets*, 6, 385-295.
- Malliaris, M., & Salchenberger, L. (1993). A neural network model for estimating option prices. *Journal of Applied Intelligence*, 3, 193-206.
- Mills, T. C. (1990). Non-linear time series models in economics. *Journal of Economic Surveys*, 5, 215-241.
- Motiwalla, L., & Wahab, M. (2000). Predictable variation and profitable trading of US equities: A trading simulation using neural networks. *Computer & Operations Research*, 27, 1111-1129.
- Nelson, M., Hill, T., Remus, W., & O'Connor, M. (1999). Time series forecasting using neural networks: Should the data be deseasonalized first? *Journal of Forecasting*, 18, 359-367.
- Pantazopoulos, K. N., Tsoukalas, L. H., Bourbakis, N. G., Brun, M. J., & Houstis, E. N. (1998). Financial prediction and trading strategies using neurofuzzy approaches. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 28, 520-530.
- Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33, 1065-1076.
- Pesaran, M. H., & Timmermann, A. (1992). A simple nonparametric test of predictive performance. *Journal of Business & Economic Statistics*, 10, 461-465.
- Pesaran, M. H., & Timmermann, A. (1995). Predictability of stock returns: Robustness and economic significance. *Journal of Finance*, 50, 1201-1227.
- Peterson, G. E., St Clair, D. C., Aylward, S. R., & Bond, W. E. (1995). Using Taguchi's method of experimental design to control errors in layered perceptrons. *IEEE Transactions on Neural Networks*, 6, 949-961.
- Poddig, T., & Rehugler, H. (1996). A world of integrated financial markets using artificial neural networks. *Neurocomputing*, 10, 251-273.
- Priestley, M. B. (1988). *Non-linear and non-stationary time series analysis*. London: Academic Press.
- Qi, M., & Maddala, G. S. (1999). Economic factors and the stock market: A new perspective. *Journal of Forecasting*, 18, 151-166.
- Quinlan, J. (1993). *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufmann.
- Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks* (pp. 586-591). San Francisco.
- Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition*. Cambridge, MA: The MIT Press.
- Schwert, W. (1990). Stock returns and real activity: A century of evidence. *Journal of Finance*, 45, 1237-1257.

- Specht, D. F. (1990). Probabilistic neural networks. *Neural Networks*, 3, 109-118.
- Specht, D. F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks*, 2, 568-576.
- Swales, G. S., & Yoon, Y. (1992). Applying artificial neural networks to investment analysis. *Financial Analysts Journal*, 48, 78-80.
- Thawornwong, S., Enke, D., & Dagli, C. (2001, November). *Neural network models for classifying the direction of excess stock return*. Paper presented at the 32nd Annual Meeting of the Decision Sciences Institute, San Francisco, CA.
- Vellido, A., Lisboa, P. J. G., & Vaughan, J. (1999). Neural networks in business: A survey of application (1992-1998). *Expert Systems with Applications*, 17, 51-70.
- Wasserman, P. D. (1993). *Advanced methods in neural computing*. New York: Van Nostrand Reinhold.
- Wood, D., & Dasgupta, B. (1996). Classifying trend movements in the MSCI U.S.A. capital market index — A comparison of regression, ARIMA, and neural network methods. *Computers & Operations Research*, 23, 611-622.
- Wu, Y., & Zhang, H. (1997). Forward premiums as unbiased predictors of future currency depreciation: A non-parametric analysis. *Journal of International Money and Finance*, 16, 609-623.

Appendix

- SP* Nominal Standard & Poor's 500 index at the close of the last trading day of each month. Source: Commodity Systems, Inc. (CSI).
- DIV* Nominal dividends per share for the S&P 500 portfolio paid during the month. Source: Annual dividend record / Standard & Poor's Corporation.
- TI* Annualized average of bid and ask yields on 1-month T-bill rate on the last trading day of the month. It refers to the shortest maturity T-bills not less than 1 month in maturity. Source: CRSP tapes. The Fama risk-free-rate files.
- TIH* Monthly holding-period return on 1-month T-bill rate on the last trading day of the month, calculated as $TI/12$.
- R* Nominal stock returns on the S&P 500 portfolio, calculated as $R_t = (SP_t - SP_{t-1})/SP_{t-1}$.
- ER* Excess stock returns on the S&P 500 portfolio, calculated as $ER_t = R_t - TIH_{t-1}$.
- DY* Dividend yield on the S&P 500 portfolio, calculated as $DY_t = DIV_t/SP_t$.
- T3* 3-month T-bill rate, secondary market, averages of business days, discount basis. Source: H.15 Release — Federal Reserve Board of Governors.
- T6* 6-month T-bill rate, secondary market, averages of business days, discount basis. Source: H.15 Release — Federal Reserve Board of Governors.

- T12* 1-year T-bill rate, secondary market, averages of business days, discount basis. Source: H.15 Release — Federal Reserve Board of Governors.
- T60* 5-year T-bill constant maturity rate, secondary market, averages of business days. Source: H.15 Release — Federal Reserve Board of Governors.
- T120* 10-year T-bill constant-maturity rate, secondary market, averages of business days. Source: H.15 Release — Federal Reserve Board of Governors.
- CD1* 1-month certificate-of-deposit rate, averages of business days. Source: H.15 Release — Federal Reserve Board of Governors.
- CD3* 3-month certificate-of-deposit rate, averages of business days. Source: H.15 Release — Federal Reserve Board of Governors.
- CD6* 6-month certificate-of-deposit rate, averages of business days. Source: H.15 Release — Federal Reserve Board of Governors.
- AAA* Moody's seasoned Aaa corporate-bond yield, averages of business days. Source: The Federal Reserve Bank of St. Louis.
- BAA* Moody's seasoned Baa corporate-bond yield, averages of business days. Source: The Federal Reserve Bank of St. Louis.
- PP* Producer Price Index: Finished Goods. Source: U.S. Department of Labor, Bureau of Labor Statistics.
- IP* Industrial Production Index: Market Groups and Industry Groups. Source: G.17 Statistical Release — Federal Reserve Statistical Release.
- CP* Consumer Price Index: CPI for All Urban Consumers. Source: U.S. Department of Labor, Bureau of Labor Statistics.
- M1* M1 Money Stock. Source: H.6 Release — Federal Reserve Board of Governors.
- TE1* Term spread between *T120* and *T1*, calculated as $TE1 = T120 - T1$.
- TE2* Term spread between *T120* and *T3*, calculated as $TE2 = T120 - T3$.
- TE3* Term spread between *T120* and *T6*, calculated as $TE3 = T120 - T6$.
- TE4* Term spread between *T120* and *T12*, calculated as $TE4 = T120 - T12$.
- TE5* Term spread between *T3* and *T1*, calculated as $TE5 = T3 - T1$.
- TE6* Term spread between *T6* and *T1*, calculated as $TE6 = T6 - T1$.
- DE1* Default spread between *BAA* and *AAA*, calculated as $DE1 = BAA - AAA$.
- DE2* Default spread between *BAA* and *T120*, calculated as $DE2 = BAA - T120$.
- DE3* Default spread between *BAA* and *T12*, calculated as $DE3 = BAA - T12$.
- DE4* Default spread between *BAA* and *T6*, calculated as $DE4 = BAA - T6$.
- DE5* Default spread between *BAA* and *T3*, calculated as $DE5 = BAA - T3$.
- DE6* Default spread between *BAA* and *T1*, calculated as $DE6 = BAA - T1$.
- DE7* Default spread between *CD6* and *T6*, calculated as $DE7 = CD6 - T6$.

Chapter IV

Hybrid-Learning Methods for Stock Index Modeling

Yuehui Chen, Jinan University, China

Ajith Abraham, Chung-Ang University, Republic of Korea

Abstract

The use of intelligent systems for stock market prediction has been widely established. In this paper, we investigate how the seemingly chaotic behavior of stock markets could be well represented using several connectionist paradigms and soft computing techniques. To demonstrate the different techniques, we consider the Nasdaq-100 index of Nasdaq Stock MarketSM and the S&P CNX NIFTY stock index. We analyze 7-year Nasdaq 100 main-index values and 4-year NIFTY index values. This chapter investigates the development of novel, reliable, and efficient techniques to model the seemingly chaotic behavior of stock markets. We consider the flexible neural tree algorithm, a wavelet neural network, local linear wavelet neural network, and finally a feed-forward artificial neural network. The particle-swarm-optimization algorithm optimizes the parameters of the different techniques. This paper briefly explains how the different learning paradigms could be formulated using various methods and then investigates whether they can provide the required level of performance — in other

words, whether they are sufficiently good and robust so as to provide a reliable forecast model for stock market indices. Experiment results reveal that all the models considered could represent the stock indices behavior very accurately.

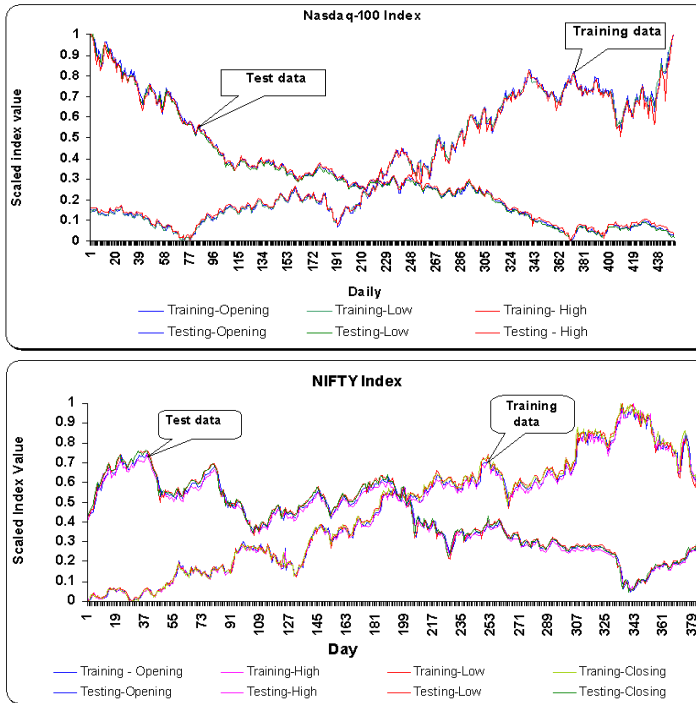
Introduction

Prediction of stocks is generally believed to be a very difficult task — it behaves like a random walk process and time varying. The obvious complexity of the problem paves the way for the importance of intelligent prediction paradigms (Abraham, Nath, & Mahanti, 2001). During the last decade, stocks and futures traders have come to rely upon various types of intelligent systems to make trading decisions (Abraham, Philip, & Saratchandran, 2003; Chan & Liu, 2002; Francis, Tay, & Cao, 2002; Leigh, Modani, Purvis, & Roberts, 2002; Leigh, Purvis, & Ragusa, 2002; Oh & Kim, 2002; Quah & Srinivasan, 1999; Wang, 2002). Several intelligent systems have in recent years been developed for modeling expertise, decision support, and complicated automation tasks (Berkeley, 1997; Bischi & Valori, 2000; Cios, 2001; Kim & Han, 2000; Koulouriotis, Diakoulakis, & Emiris, 2001; Lebaron, 2001; Palma-dos-Reis & Zahedi, 1999; Wuthrich et al., 1998). In this chapter, we analyse the seemingly chaotic behavior of two well-known stock indices namely the Nasdaq-100 index of NasdaqSM and the S&P CNX NIFTY stock index.

The Nasdaq-100 index reflects Nasdaq's largest companies across major industry groups, including computer hardware and software, telecommunications, retail/wholesale trade, and biotechnology. The Nasdaq-100 index is a modified capitalization-weighted index, designed to limit domination of the Index by a few large stocks while generally retaining the capitalization ranking of companies. Through an investment in Nasdaq-100 index tracking stock, investors can participate in the collective performance of many of the Nasdaq stocks that are often in the news or have become household names. Similarly, S&P CNX NIFTY is a well-diversified 50-stock index accounting for 25 sectors of the economy. It is used for a variety of purposes such as benchmarking fund portfolios, index-based derivatives, and index funds. The CNX indices are computed using the market capitalization weighted method, wherein the level of the index reflects the total market value of all the stocks in the index relative to a particular base period. The method also takes into account constituent changes in the index and importantly corporate actions such as stock splits, rights, and so on, without affecting the index value.

Our research investigates the performance analysis of four different connectionist paradigms for modeling the Nasdaq-100 and NIFTY stock market indices. We consider the Flexible Neural Tree (FNT) algorithm (Chen, Yang, and Dong, 2004), a Wavelet Neural Network (WNN), Local Linear Wavelet Neural Network (LLWNN) (Chen et al., 2006) and finally a feed-forward Neural Network (ANN) (Chen et al., 2004). The particle-swarm-optimization algorithm optimizes the parameters of the different techniques (Kennedy & Eberhart, 1995). We analysed the Nasdaq-100 index value from 11 January 1995 to 11 January 2002 and the NIFTY index from 01 January 1998 to 03 December 2001. For both indices, we divided the entire data into roughly two equal halves. No special rules were used to select the training set other than ensuring a reasonable representation of the

Figure 1. (a) Training and test data sets for the Nasdaq-100 index and (b) the NIFTY index



parameter space of the problem domain (Abraham et al., 2003). The complexity of the training and test data sets for both indices is depicted in Figure 1. In the section entitled “Hybrid-learning Models,” we briefly describe the different learning algorithms. This section is followed by the “Experimentation Setup and Results” section. This is, in turn, followed by the “Conclusions” section.

Particle-Swarm-Optimization (PSO) Algorithm

The PSO conducts searches using a population of particles that correspond to individuals in an Evolutionary Algorithm (EA). Initially, a population of particles is randomly generated. Each particle represents a potential solution and has a position represented by a position vector x_i . A swarm of particles moves through the problem space, with the moving velocity of each particle represented by a velocity vector v_i . At each time step, a function f_i — representing a quality measure — is calculated by using x_i as input. Each

particle keeps track of its own *best* position, which is associated with the best fitness it has achieved so far in a vector p_i . Furthermore, the *best* position among all the particles obtained so far in the population is kept track of as p_g . In addition to this global version, another version of PSO keeps track of the *best* position among all the topological neighbors of a particle. At each time step t , by using the individual best position, $p_i(t)$, and the global best position, $p_g(t)$, a new velocity for particle i is updated by:

$$v_i(t+1) = v_i(t) + c_1\phi_1(p_i(t) - x_i(t)) + c_2\phi_2(p_g(t) - x_i(t)) \quad (1)$$

where c_1 and c_2 are positive constants and ϕ_1 and ϕ_2 are uniformly distributed random numbers in $[0, 1]$. The term c_i is limited to the range of $\pm V_{\max}$ (if the velocity violates this limit, it is set to its proper limit). Changing velocity this way enables the particle i to search around both its individual best position, p_i , and global best position, p_g . Based on the updated velocities, each particle changes its position according to:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

The PSO algorithm is employed to optimize the parameter vectors of FNT, ANN, and WNN.

Hybrid-Learning Models

Flexible Neural Tree Model

In this research, a tree-structured encoding method with specific instruction set is selected for representing a FNT model (Chen et al., 2004, 2005).

Flexible Neuron Instructor and FNT Model

The function set F and terminal instruction set T used for generating a FNT model are described as follows:

$$S = F \cup T = \{+,_2, +_3, \dots, +_N\} \cup \{x_1, x_2, \dots, x_n\} \quad (3)$$

where $+,_i$ ($i = 2, 3, \dots, N$) denote nonleaf nodes' instructions and taking i arguments. x_1, x_2, \dots, x_n are leaf nodes' instructions and taking no other arguments. The output of a nonleaf

node is calculated as a flexible neuron model (see Figure 2). From this point of view, the instruction $+$ is also called a flexible neuron operator with i inputs.

In the construction process of a neural tree, if a nonterminal instruction, that is, $+$ ($i = 2, 3, \dots, N$) is selected, i real values are randomly generated and used for representing the connection strength between the node $+$ and its children. In addition, two adjustable parameters a_i and b_i are randomly created as flexible activation function parameters.

For developing the FNT model, the following flexible activation function is used:

$$f(a_i, b_i; x) = \exp\left(-\frac{(x - a_i)^2}{b_i^2}\right) \tag{4}$$

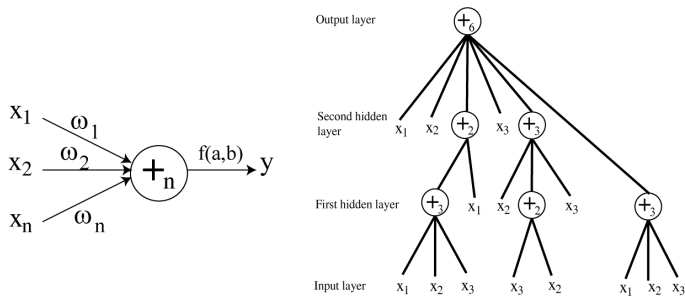
The output of a flexible neuron $+$ can be calculated as follows. The total excitation of $+$ is:

$$net_n = \sum_{j=1}^n w_j x_j \tag{5}$$

where $x_j (j = 1, 2, \dots, n)$ are the inputs to node $+$. The output of the node $+$ is then calculated by:

$$out_n = f(a_n, b_n, net_n) = \exp\left(-\frac{(net_n - a_n)^2}{b_n^2}\right) \tag{6}$$

Figure 2. A flexible neuron operator (left), and a typical representation of the FNT with function instruction set $F = \{+, +_2, +_3, \dots, +_6\}$, and terminal instruction set $T = \{x_1, x_2, x_3\}$



A typical flexible neuron operator and a neural tree model are illustrated in Figure 2. The overall output of the flexible neural tree can be recursively computed from left to right by the depth-first method.

Optimization of the FNT Model

The optimization of FNT includes both tree-structure and parameter optimization. Finding an optimal or near-optimal neural tree is formulated as a product of evolution. A number of neural tree variation operators are developed as follows:

- Mutation

Four different mutation operators were employed to generate offspring from the parents. These mutation operators are as follows:

- (1) Changing one terminal node: Randomly select one terminal node in the neural tree and replace it with another terminal node.
- (2) Changing all the terminal nodes: Select each and every terminal node in the neural tree and replace it with another terminal node.
- (3) Growing: Select a random leaf in the hidden layer of the neural tree and replace it with a newly generated subtree.
- (4) Pruning: Randomly select a function node in the neural tree and replace it with a terminal node.

The neural tree operators were applied to each of the parents to generate an offspring using the following steps:

- (a) A Poisson random number N , with mean λ , was generated.
- (b) N random mutation operators were uniformly selected with replacement from the previous four-mutation operator set.
- (c) These N mutation operators were applied in sequence one after the other to the parents to get the offspring.

- Crossover

Select two neural trees at random and select one nonterminal node in the hidden layer for each neural tree randomly, then swap the selected subtree. The crossover operator is implemented with a predefined probability of 0.3 in this study.

- Selection

Evolutionary-programming (EP) tournament selection was applied to select the parents for the next generation. Pairwise comparison is conducted for the union of μ parents and μ offspring. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is no smaller than the opponent's, it is selected. Then select μ individuals from parents and offspring that have most wins to form the next generation.

- Parameter Optimization by PSO

Parameter optimization is achieved by the PSO algorithm as described in the “The Particle-swarm-optimization (PSO) Algorithm” section. In this stage, the FNT architecture is fixed, as the best tree developed by the end of run of the structured search. The parameters (weights and flexible activation-function parameters) encoded in the best tree formulate a particle. The PSO algorithm works as follows:

- (a) An initial population is randomly generated. The learning parameters c_1 and c_2 in PSO should be assigned in advance.
- (b) The objective function value is calculated for each particle.
- (c) Modification of search point — the current search point of each particle is changed using Equations 1 and 2.
- (d) If the maximum number of generations is reached or no better parameter vector is found for a significantly long time (~ 100 steps), then stop, otherwise go to step (b).

The Artificial Neural Network (ANN) Model

A neural network classifier trained using the PSO algorithm with flexible bipolar sigmoid activation functions at hidden layer was constructed for the stock data. Before describing the details of the algorithm for training the ANN classifier, the issue of coding needs to be addressed. Coding concerns the way the weights and the flexible activation-function parameters of the ANN are represented by individuals or particles. A floating-point coding scheme is adopted here. For neural network (NN) coding, suppose there are M nodes in the hidden layer and one node in the output layer and n input variables, then the number of total weights is $n \times M + M \times 1$, the number of thresholds is $M + 1$ and the number of flexible activation-function parameters is $M + 1$, therefore the total number of free parameters in the ANN to be coded is $n \times M + M + 2(M + 1)$. These parameters are coded into an individual or particle orderly. The simple proposed training algorithm for a neural network is the same as the PSO algorithm.

The WNN-Prediction Model

In terms of wavelet transformation theory, wavelets in the following form:

$$\psi = \{\psi_i = \left| a_i^{-\frac{1}{2}} \right| \varphi\left(\frac{x-b_i}{a_i}\right) : a_i, b_i \in R, i \in Z\} \quad (7)$$

$$x = (x_1, x_2, \dots, x_n), \quad a_i = (a_{i1}, a_{i2}, \dots, a_{in}), \quad b_i = (b_{i1}, b_{i2}, \dots, b_{in})$$

are a family of functions generated from one single function $\varphi(x)$ by the operation of dilation and translation. $\varphi(x)$, which is localized in both the time space and the frequency space, is called a mother wavelet and the parameters a_i and b_i are named the scale and translation parameters, respectively.

In the standard form of a wavelet neural network, output is given by:

$$f(x) = \sum_{i=1}^M \omega_i \psi_i(x) = \sum_{i=1}^M \omega_i \left| a_i^{-\frac{1}{2}} \right| \varphi\left(\frac{x-b_i}{a_i}\right) \quad (8)$$

where ψ_i is the wavelet activation function of i -th unit of the hidden layer and ω_i is the weight connecting the i -th unit of the hidden layer to the output-layer unit. Note that for the n -dimensional input space, the multivariate wavelet-basis function can be calculated by the tensor product of n single wavelet-basis functions as follows:

$$\varphi(x) = \prod_{i=1}^n \varphi(x_i) \quad (9)$$

Before describing details of the PSO algorithm for training WNNs, the issue of coding needs to be addressed. Coding concerns the way the weights, dilation, and translation parameters of WNNs are represented by individuals or particles. A floating-point coding scheme is adopted here. For WNN coding, suppose there are M nodes in the hidden layer and n input variables, then the total number of parameters to be coded is $(2n + 1)M$. The coding of a WNN into an individual or particle is as follows:

$$| a_{11} b_{11} \cdots a_{1n} b_{1n} \omega_1 | a_{21} b_{21} \cdots a_{2n} b_{2n} \omega_2 | \cdots | a_{n1} b_{n1} \cdots a_{nm} b_{nm} \omega_n |$$

The simple proposed training algorithm for a WNN is as follows:

Step 1: An initial population is randomly generated. The learning parameters, such as c_1 , c_2 in PSO should be assigned in advance.

Step 2: Parameter optimization with PSO algorithm.

Step 3: if the maximum number of generations is reached or no better parameter vector is found for a significantly long time (~ 100 steps), then go to *Step 4*; otherwise go to *Step 2*.

Step 4: Parameter optimization with gradient-descent algorithm.

Step 5: If a satisfactory solution is found then stop; otherwise go to *Step 4*.

The Local Linear WNN Prediction Model

An intrinsic feature of basis-function networks is the localized activation of the hidden-layer units, so that the connection weights associated with the units can be viewed as locally accurate piecewise constant models whose validity for any given input is indicated by the activation functions. Compared to the multilayer perceptron neural network, this local capacity provides some advantages, such as learning efficiency and structure transparency. However, the problem of basis-function networks requires some special attention. Due to the crudeness of the local approximation, a large number of basis-function units have to be employed to approximate a given system. A shortcoming of the wavelet neural network is that for higher dimensional problems many hidden-layer units are needed.

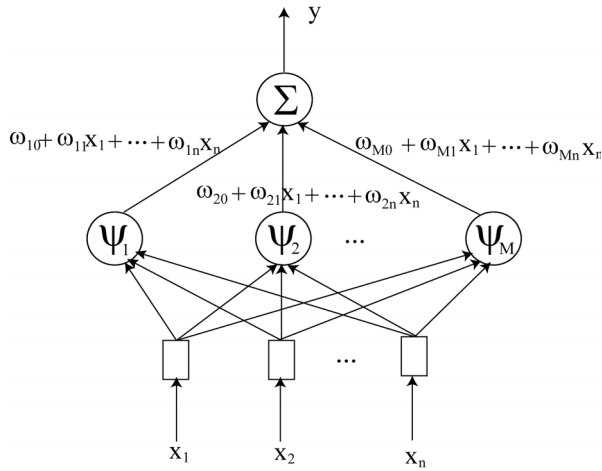
In order to take advantage of the local capacity of the wavelet-basis functions while not having too many hidden units, here we propose an alternative type of WNN. The architecture of the proposed local linear WNN (LLWNN) is shown in Figure 3. Its output in the output layer is given by:

$$y = \sum_{i=1}^M (\omega_{i0} + \omega_{i1}x_1 + \cdots + \omega_{in}x_n) \psi_i(x) = \sum_{i=1}^M (\omega_{i0} + \omega_{i1}x_1 + \cdots + \omega_{in}x_n) |a_i|^{-\frac{1}{2}} \varphi\left(\frac{x-b_i}{a_i}\right) \quad (10)$$

where $x = (x_1, x_2, \dots, x_n)$. Instead of the straightforward weight ω_i (piecewise constant model), a linear model:

$$v_i = \omega_{i0} + \omega_{i1}x_1 + \cdots + \omega_{in}x_n \quad (11)$$

Figure 3. Architecture of a local linear wavelet neural network



is introduced. The activities of the linear model v_i ($i = 1, 2, \dots, M$) are determined by the associated locally active wavelet function $\psi_i(x)$ ($i = 1, 2, \dots, M$), thus v_i is only locally significant. The motivations for introducing local linear models into a WNN are as follows: (1) Local-linear models have been studied in some neurofuzzy systems (Abraham, 2001) and offer good performances; and (2) Local-linear models should provide a more parsimonious interpolation in high-dimension spaces when modeling samples are sparse. The scale and translation parameters and local-linear-model parameters are randomly initialized at the beginning and are optimized by the PSO algorithm.

Experiment Setup and Results

We considered 7-year stock data for the Nasdaq-100 Index and 4-year for the NIFTY index. Our target was to develop efficient forecast models that could predict the index value of the following trading day based on the opening, closing, and maximum values on any given day. The training and test patterns for both indices (scaled values) are illustrated in Figure 1. We used the same training- and test-data sets to evaluate the different connectionist models. More details are reported in the following sections. Experiments were carried out on a Pentium IV, 2.8 GHz Machine with 512 MB RAM and the programs implemented in C/C++. Test data was presented to the trained connectionist models, and the output from the network compared with the actual index values in the time series.

The assessment of the prediction performance of the different connectionist paradigms were done by quantifying the prediction obtained on an independent data set. The root-mean-squared error (*RMSE*), maximum-absolute-percentage error (*MAP*), mean-absolute-percentage error (*MAPE*), and correlation coefficient (*CC*) were used to study the performance of the trained forecasting model for the test data.

MAP is defined as follows:

$$MAP = \max_i \frac{|P_{\text{actual},i} - P_{\text{predicted},i}|}{P_{\text{predicted},i}} \cdot 100 \quad (12)$$

where $P_{\text{actual},i}$ is the actual index value on day i and $P_{\text{predicted},i}$ is the forecast value of the index on that day. Similarly *MAPE* is given as:

$$MAPE = \frac{1}{N} \sum_{i=1}^N \frac{|P_{\text{actual},i} - P_{\text{predicted},i}|}{P_{\text{actual},i}} \cdot 100 \quad (13)$$

where N represents the total number of days.

- FNT Algorithm

We used the instruction set $S = \{+_2, +_3, \dots, +_{10}, x_0, x_1, x_2\}$ modeling the Nasdaq-100 index and instruction set $S = \{+_2, +_3, \dots, +_{10}, x_0, x_1, x_2, x_3, x_4\}$ modeling the NIFTY index. We used the flexible activation function of Equation 4 for the hidden neurons. Training was terminated after 80 epochs on each dataset.

- NN-PSO Training

A feed-forward neural network with three input nodes and a single hidden layer consisting of 10 neurons was used for modeling the Nasdaq-100 index. A feed-forward neural network with five input nodes and a single hidden layer consisting of 10 neurons was used for modeling the NIFTY index. Training was terminated after 3000 epochs on each dataset.

- WNN-PSO

A WNN with three input nodes and a single hidden layer consisting of 10 neurons was used for modeling the Nasdaq-100 index. A WNN with five input nodes and a single

hidden layer consisting of 10 neurons was used for modeling the NIFTY index. Training was terminated after 4000 epochs on each dataset.

- LLWNN-PSO

A LLWNN with three input nodes and a hidden layer consisting of five neurons for modeling Nasdaq-100 index. A LLWNN with five input nodes and a single hidden layer consisting of five neurons for modeling NIFTY index. Training was terminated after 4500 epochs on each dataset.

Figure 4. Test results showing the performance of the different methods for modeling the Nasdaq-100 index

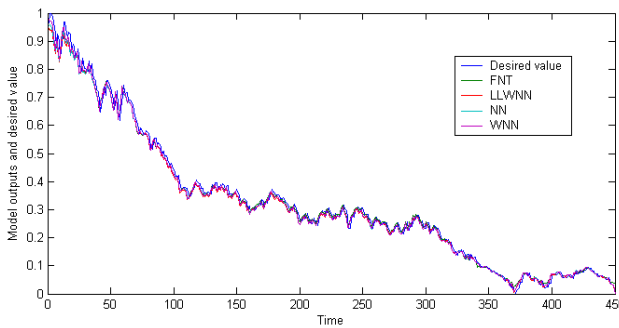


Figure 5. Test results showing the performance of the different methods for modeling the NIFTY index

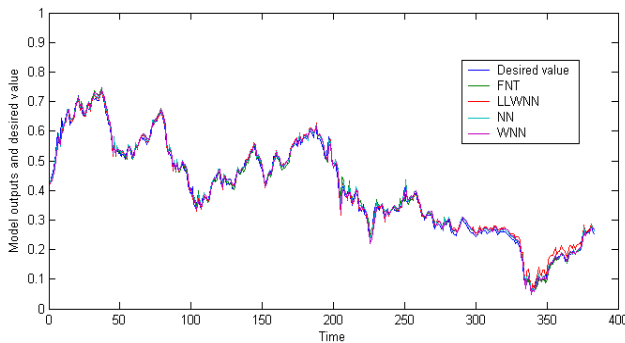


Table 1. Empirical comparison of RMSE results for four learning methods

	FNT	NN-PSO	WNN-PSO	LLWNN-PSO
Training results				
Nasdaq-100	0.02598	0.02573	0.02586	0.02551
NIFTY	0.01847	0.01729	0.01829	0.01691
Testing results				
Nasdaq-100	0.01882	0.01864	0.01789	0.01968
NIFTY	0.01428	0.01326	0.01426	0.01564

Table 2. Statistical analysis of four learning methods (test data)

	FNT	NN-PSO	WNN-PSO	LLWNN-PSO
Nasdaq-100				
CC	0.997579	0.997704	0.997721	0.997623
MAP	98.107	141.363	152.754	230.514
MAPE	6.205	6.528	6.570	6.952
NIFTY				
CC	0.996298	0.997079	0.996399	0.996291
MAP	39.987	27.257	39.671	30.814
MAPE	3.328	3.092	3.408	4.146

- Performance and Results Achieved

Table 1 summarizes the training and test results achieved for the two stock indices using the four different approaches. The statistical analysis of the four learning methods is depicted in Table 2. Figures 4 and 5 depict the test results for the 1-day-ahead prediction of Nasdaq-100 index and NIFTY index respectively.

Conclusion

In this chapter, we have demonstrated how the chaotic behavior of stock indices could be well-represented by different hybrid learning paradigms. Empirical results on the two data sets using four different learning models clearly reveal the efficiency of the proposed techniques. In terms of RMSE values, for the Nasdaq-100 index, WNN performed marginally better than the other models and for the NIFTY index, the NN approach gave the lowest generalization RMSE values. For both data sets, LLWNN had the lowest training error. For the Nasdaq-100 index (test data), WNN had the highest CC, but the lowest values of MAPE and MAP were achieved by using the FNT model. The highest CC together with the best MAPE/MAP values for the NIFTY index were achieved using the NN trained using the PSO model. A low MAP value is a crucial indicator for evaluating the stability of a market under unforeseen fluctuations. In the present example, the predictability ensures that the decrease in trade is only a temporary cyclic variation that is perfectly under control.

Our research was to predict the share price for the following trading day based on the opening, closing, and maximum values on any given day. Our experimental results

indicate that the most prominent parameters that affect share prices are their immediate opening and closing values. The fluctuations in the share market are chaotic in the sense that they heavily depend on the values of their immediate forerunning fluctuations. Long-term trends exist, but are slow variations and this information is useful for long-term investment strategies. Our study focused on short-term floor trades in which the risk is higher. However, the results of our study show that even with seemingly random fluctuations, there is an underlying deterministic feature that is directly enciphered in the opening, closing, and maximum values of the index of any day making predictability possible.

Empirical results also show that there are various advantages and disadvantages for the different techniques considered. There is little reason to expect that one can find a uniformly best learning algorithm for optimization of the performance for different stock indices. This is in accordance with the no-free-lunch theorem, which explains that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class (Macready & Wolpert, 1997). Our future research will be oriented towards determining the optimal way to combine the different learning paradigms using an ensemble approach (Maqsood, Kahn, & Abraham, 2004) so as to complement the advantages and disadvantages of the different methods considered.

Acknowledgment

This research was partially supported by the Natural Science Foundation of China under grant number 60573065, and The Provincial Science and Technology Development Program of Shandong under grant number SDSP2004-0720-03.

References

- Abraham, A. (2001). NeuroFuzzy systems: State-of-the-art modeling techniques. In J. Mira & A. Prieto (Eds.), *Proceedings of the 7th International Work Conference on Artificial and Neural Networks, Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, Granada, Spain (pp. 269-276). Germany: Springer-Verlag.
- Abraham, A., Nath, B., & Mahanti, P. K. (2001). Hybrid intelligent systems for stock market analysis. In V. N. Alexandrov, J. Dongarra, B. A. Julianno, R. S. Renner, & C. J. K. Tan (Eds.), *Computational science* (pp. 337-345). Germany: Springer-Verlag.
- Abraham, A., Philip, N. S., & Saratchandran, P. (2003). Modeling chaotic behavior of stock indices using intelligent paradigms. *International Journal of Neural, Parallel & Scientific Computations*, 11(1-2), 143-160.

- Berkeley, A. R. (1997). Nasdaq's technology floor: Its president takes stock. *IEEE Spectrum*, 34(2), 66-67.
- Bischi, G. I., & Valori, V. (2000). Nonlinear effects in a discrete-time dynamic model of a stock market. *Chaos, Solitons & Fractals*, 11(13), 2103-2121.
- Chan, W. S., & Liu, W. N. (2002). Diagnosing shocks in stock markets of Southeast Asia, Australia, and New Zealand. *Mathematics and Computers in Simulation*, 59(1-3), 223-232.
- Chen, Y., Yang, B., & Dong, J. (2004). Nonlinear system modeling via optimal design of neural trees. *International Journal of Neural Systems*, 14(2), 125-137.
- Chen, Y., Yang, B., & Dong, J. (2006). Time-series prediction using a local linear wavelet neural network. *International Journal of Neural Systems*, 69(4-6), 449-465.
- Chen, Y., Yang, B., Dong, J., & Abraham, A. (2005). Time-series forecasting using flexible neural tree model. *Information Science*, 174(3-4), 219-235.
- Cios, K. J. (2001). Data mining in finance: Advances in relational and hybrid methods. *Neurocomputing*, 36(1-4), 245-246.
- Francis, E. H., Tay, H., & Cao, L. J. (2002). Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1-4), 847-861.
- Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks* (pp. 1942-1948), Perth, Australia. Piscataway, NJ: IEEE Service Center.
- Kim, K. J., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19(2), 125-132.
- Koulouriotis, D. E., Diakoulakis, I. E., & Emiris, D. M. (2001). A fuzzy cognitive map-based stock market model: Synthesis, analysis and experimental results. In *Proceedings of the 10th IEEE International Conference on Fuzzy Systems, Vol. 1* (pp. 465-468).
- Lebaron, B. (2001). Empirical regularities from interacting long- and short-memory investors in an agent-based stock market. *IEEE Transactions on Evolutionary Computation*, 5(5), 442-455.
- Leigh, W., Modani, N., Purvis, R., & Roberts, T. (2002). Stock market trading rule discovery using technical charting heuristics. *Expert Systems with Applications*, 23(2), 155-159.
- Leigh, W., Purvis, R., & Ragusa, J. M. (2002). Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: A case study in romantic decision support. *Decision Support Systems*, 32(4), 361-377.
- Macready, W. G., & Wolpert, D. H. (1997). *The nofree lunch theorems*. *IEEE Transaction on Evolutionary Computing*, 1(1), 67-82.
- Maqsood, I., Khan, M. R., & Abraham, A. (2004). Neural network ensemble method for weather forecasting. *Neural Computing & Applications*, 13(2), 112-122.

- Nasdaq Stock MarketSM. (n.d.). Retrieved February 8, 2006, from <http://www.nasdaq.com>
- National Stock Exchange of India Limited. (n.d.). Retrieved February 8, 2006, from <http://www.nse-india.com>
- Oh, K. J., & Kim, K. J. (2002). Analyzing stock market tick data using piecewise nonlinear model. *Expert Systems with Applications*, 22(3), 249-255.
- Palma-dos-Reis, A., & Zahedi, F. (1999). Designing personalized intelligent financial decision support systems. *Decision Support Systems*, 26(1), 31-47.
- Quah, T. S., & Srinivasan, B. (1999). Improving returns on stock investment through neural network selection. *Expert Systems with Applications*, 17(4), 295-301.
- Wang, Y. F. (2002). Mining stock price using fuzzy rough set system. *Expert Systems with Applications*, 24(1), 13-23.
- Wuthrich, B., Cho, V., Leung, S., Permuntilleke, D., Sankaran, K., & Zhang, J. (1998). Daily stock market forecast from textual web data. *Proceedings IEEE International Conference on Systems, Man, and Cybernetics*, 3, 2720-2725.

Chapter V

Application of Higher-Order Neural Networks to Financial Time-Series Prediction

John Fulcher, University of Wollongong, Australia

Ming Zhang, Christopher Newport University, USA

Shuxiang Xu, University of Tasmania, Australia

Abstract

Financial time-series data is characterized by nonlinearities, discontinuities, and high-frequency multipolynomial components. Not surprisingly, conventional artificial neural networks (ANNs) have difficulty in modeling such complex data. A more appropriate approach is to apply higher-order ANNs, which are capable of extracting higher-order polynomial coefficients in the data. Moreover, since there is a one-to-one correspondence between network weights and polynomial coefficients, higher-order neural networks (HONNs) — unlike ANNs generally — can be considered open-, rather than “closed-box” solutions, and thus hold more appeal to the financial community. After developing polynomial and trigonometric HONNs (P[T]HONNs), we introduce the concept of HONN groups. The latter incorporate piecewise continuous-activation

functions and thresholds, and as a result are capable of modeling discontinuous (or piecewise-continuous) data, and what is more to any degree of accuracy. Several other PHONN variants are also described. The performance of P(T)HONN and HONN groups on representative financial time series is described (i.e., credit ratings and exchange rates). In short, HONNs offer roughly twice the performance of MLP/BP on financial time-series prediction, and HONN groups around 10% further improvement.

Financial Time Series Prediction

It is clear that there are pattern(s) underlying some time series. For example, the 11-year cycle observed in sunspot data (University of California, Irvine, 2005). Whether this is the case with financial time-series data is debatable. For instance, do underlying “forces” actually drive financial markets, and if so can their existence be deduced by observations of stock price and volume movements (Back, 2004)?

Alternatively, do so-called “market inefficiencies” exist, whereby it is possible to devise strategies to consistently “beat the market” in terms of return-on-investment (Edelman & Davy, 2004)? If this is in fact the case, then it runs counter to the so-called Efficient Markets Hypothesis, namely that the present pricing of a financial asset is a reflection of all the available information about that asset, whether this be private (insider), public, or previous pricing (if based *solely* on the latter, then this is referred to as the “weak form” of the EMH).

Market traders, by contrast, tend to base their decisions not only on the previous considerations, but also on many other factors, including hunches (intuition). Quantifying these often complex decision-making processes (expertise) is a difficult, if not impossible, task akin to the fundamental problem inherent in designing *any* Expert System. An overriding consideration is that any model (system) tends to break down in the face of singularities, such as stock market crashes (e.g., “Black Tuesday”, October 1987), war, political upheaval, business scandals, rumor, panic buying, and so on.

“Steady-state” markets, on the other hand, tend to exhibit *some* predictability, albeit minor — for example, so-called “calendar effects”: lower returns on Mondays, higher returns on the last day of the month and just prior to public holidays, higher returns in January, and so on (Kingdon, 1997).

Now, while it is possible that financial time-series data on occasion can be described by a linear function, most often it is characterized by nonlinearities, discontinuities, and high-frequency multipolynomial components.

If there *is* an underlying market model, then it has remained largely impervious to statistical (and other forms of) modeling. We can take a lead here from adaptive control systems and/or machine learning; in other words, if a system is too complex to model, try *learning* it. This is where techniques such as ANNs can play a role.

Many different techniques have been applied to financial time-series forecasting over the years, ranging from conventional, model-based, statistical approaches to more esoteric, data-driven, experimental ones (Harris & Sollis, 2003; Mills, 1993; Reinsel, 1997).

Some examples of the former are Auto Regression (AR), ARCH, Box-Jenkins (Box & Jenkins, 1976), and Kalman Filter (Harvey, 1989). Some examples of the latter are ANNs (Zhang, Patuwo, & Hu, 1998), Fuzzy Logic and variants (Sisman-Yilmaz, Alpaslan, & Jain, 2004), Evolutionary Algorithms (Allen & Karjalainen, 1999; Chen, 2002), Genetic Programming (Chen, 2002; Iba & Sasaki, 1999), Support Vector Machines (Edelman & Davy, 2004; Tay & Cao, 2001), Independent Component Analysis (Back, 2004), and other so-called (often biologically inspired) “soft computing” techniques (Kingdon, 1997). We focus on ANNs in this chapter, more specifically on *higher-order* neural networks, for reasons that we shall elaborate upon shortly.

Artificial Neural Networks (ANNs)

When people speak of ANNs, they are most likely referring to feed-forward Multilayer Perceptrons (MLPs), which employ the backpropagation (BP) training algorithm (e.g., Lapedes & Farber, 1987; Refenes, 1994; Schoneberg, 1990). Following the lead of the M-competition for different forecasting techniques (Makridakis, Anderson, Carbone, Fildes, Hibon, Lewandowski, et al., 1982), in which such ANNs compared favorably with the Box-Jenkins method, Weigand and Gershenfeld (1993) compared nonlinear forecasting techniques on a number of different time series, one of which being currency exchange rate. ANNs, along with state-space reconstruction techniques, fared well in this more recent comparative study.

At first sight, it would appear that MLP/BPs should perform reasonably well at financial time-series forecasting, since they are known to excel at (static) pattern recognition and/or classification; in this particular case, the patterns of interest are simply different time-shifted samples taken from the same data series.

Now Hornik (1991) has shown that an MLP with an arbitrary bounded nonconstant activation is capable of universal approximation. More specifically, a single hidden layer MLP/BP can approximate arbitrarily closely any suitably smooth function (Hecht-Nielsen, 1987; Hornik, Stinchcombe, & White, 1989). Furthermore, this approximation improves as the number of nodes in the hidden layer increases. In other words, a suitable network can always be found.

A similar but more extended result for learning conditional probability distributions was found by Allen and Taylor (1994). Here, two network layers are required in order to produce a smooth limit when the stochastic series (such as financial data) being modeled becomes noise free.

During learning, the outputs of a supervised neural network come to approximate the target values given the inputs in the training set. This ability may be useful in itself, but more often the purpose of using a neural net is to generalize — in other words, to have the network outputs approximate target values given inputs that are *not* in the training set.

Generally speaking, there are three conditions that are typically necessary — although not sufficient — for good generalization.

The first necessary condition is that the network inputs contain sufficient information pertaining to the target, so that there exists a mathematical function relating correct outputs to inputs with the desired degree of accuracy (Caudill & Butler, 1990).

The second necessary condition is that the function we are attempting to learn (relating inputs to desired outputs) be, in some sense, smooth (Devroye, Györfi, & Lugosi, 1996; Plotkin, 1993). In other words, small changes in inputs should produce small changes in outputs, at least most of the time. For continuous inputs and targets, function smoothness implies continuity and restrictions on the first derivative over most of the input space. Now some neural networks — including the present authors' HONN models — are able to learn discontinuities, provided the function consists of a finite number of continuous pieces. Conversely, very nonsmooth functions (such as those produced by pseudorandom number generators and encryption algorithms) are not able to be generalized by standard neural networks.

The third necessary condition for good generalization is that the training exemplars constitute a sufficiently large and representative subset (“sample” in statistics terminology) of the set of all cases we want to generalize to (the “population” in statistics terminology) (Wolpert, 1996a, 1996b). The importance of this condition is related to the fact that there are, generally speaking, two different types of generalization: interpolation and extrapolation. Interpolation applies to cases that are more or less surrounded by nearby training cases; everything else is extrapolation. In particular, cases that are outside the range of the training data require extrapolation. Cases inside large “holes” in the training data may also effectively require extrapolation. Interpolation can often be performed reliably, but extrapolation is notoriously unreliable. Hence, it is important to have sufficient training data to avoid the need for extrapolation. Methods for selecting good training sets are discussed in numerous statistical textbooks on sample surveys and experimental design (e.g., Diamond & Jeffries, 2001).

Despite the universal approximation capability of MLP/BP networks, their performance is limited when applied to financial time-series modeling and/or prediction (forecasting). This is due in part to two limitations of feed-forward ANNs, namely (Zhang, Xu, & Fulcher, 2002):

1. Their activation functions have fixed parameters only (e.g., sigmoid, radial-basis function, and so on), and
2. They are capable of continuous function approximation only; MLPs are unable to handle discontinuous and/or piecewise-continuous (economic) time-series data.

Networks with *adaptive* activation functions seem to provide better fitting properties than classical architectures with *fixed* activation-function neurons. Vecchi, Piazza, and Uncini (1998) studied the properties of a feed-forward neural network (FNN) which was able to adapt its activation function by varying the control points of a Catmull-Rom cubic spline. Their simulations confirmed that the special learning mechanism allows us to use the network's free parameters in a very effective way. In Chen and Chang (1996), real variables a (gain) and b (slope) in the generalized sigmoid activation function were adjusted during the learning process. They showed that from the perspective of static

and dynamical system modeling, use of adaptive sigmoids (in other words, sigmoids with free parameters) leads to improved data modeling compared with classical FNNs. Campolucci, Capparelli, Guarnieri, Piazza, and Uncini (1996) built an adaptive activation function as a piecewise approximation with suitable cubic splines. This function had arbitrary shape and allowed the overall size of the neural network to be reduced, trading connection complexity against activation function complexity. Several other authors (Hu & Shao, 1992; Yamada & Yabuta, 1992) have also studied the properties of neural networks that utilize adaptive activation functions.

In short, some researchers have devoted their attention to more sophisticated, alternative ANN models. One natural extension is to incorporate unit time delays (memory elements) to turn the MLP/BP into a recurrent network, in order to recognize (classify) dynamic rather than static input patterns. Alternatively, replication of network nodes and weights across time leads to time-delay neural networks, in which the layer inputs are time-shifted versions from the same time-series data. Such attempts to incorporate temporal units into an ANN have not usually led to significant improvements in financial time-series modeling/predicting performance though.

Higher-Order Neural Networks (HONNs)

Traditional areas in which ANNs are known to excel are pattern recognition, pattern matching, and mathematical function approximation (nonlinear regression). However, they suffer from several well-known limitations. They can often become stuck in local, rather than global minima, as well as taking unacceptably long times to converge in practice. Of particular concern, especially from the perspective of financial time-series prediction, is their inability to handle nonsmooth, discontinuous training data and complex mappings (associations). Another limitation of ANNs is their “black box” nature — meaning that explanations (reasons) for their decisions are not immediately obvious, unlike some other techniques, such as decision trees.

This then is the motivation for developing higher-order neural networks (HONNs).

Background on HONNs

The term “higher-order” neural network can mean different things to different people, ranging from a description of the neuron activation function to preprocessing of the neuron inputs, signifying connections to more than one layer or just ANN functionality (in other words, their ability to extract higher-order correlations from the training data). In this chapter, we use “HONN” to refer to the incorporation of a range of neuron types: linear, power, multiplicative, sigmoid, and logarithmic (see Figure 3).

HONNs have traditionally been characterized as those in which the input to a computational neuron is a weighted sum of the products of its inputs (Lee et al., 1986). Such neurons are sometimes called higher-order processing units (HPUs) (Lippmann, 1989). It has been established that HONNs can successfully perform invariant pattern recognition (Psaltis, Park, & Hong, 1988; Reid, Spirkovska, & Ochoa, 1989; Wood & Shawe-

Taylor, 1996). Giles and Maxwell (1987) showed that HONNs have impressive computational, storage, and learning capabilities. Redding, Kowalski and Downs (1993) proved that HONNs were at least as powerful as any other (similar order) FNN. Kosmatopoulos, Polycarpou, Christodoulou, & Ioannou (1995) studied the approximation and learning properties of one class of recurrent HONNs and applied these architectures to the identification of dynamical systems. Thimm and Fiesler (1997) proposed a suitable initialization method for HONNs and compared this with FNN-weight initialization.

First-order neural networks can be formulated as follows, assuming simple McCullough-and-Pitts-type neurons (Giles & Maxwell, 1987):

$$y_i(x) = f \left[\sum_j^N W(i, j)x(j) \right] \tag{1}$$

where $\{\mathbf{x}(\mathbf{j})\}$ = an N-element input vector, $W(i,j)$ = adaptable weights from all other neurons to neuron-i, and f = neuron threshold function (e.g., sigmoid). Such neurons are said to be linear, since they are only capable of capturing first-order correlations in the training data. In this sense, they can be likened to Least Mean Squared or Delta learning, as used in ADALINE. It is well known that Rosenblatt’s original (two-layer) perceptron was only capable of classifying linearly separable training data. It was not until the emergence of *Multilayer* Perceptrons (which incorporated nonlinear activation functions, such as sigmoid) that more complex (*nonlinear*) data could be discriminated.

Higher-order correlations in the training data require more complex neuron activation functions, characterized as follows (Barron, Gilstrap, & Shrier, 1987; Giles & Maxwell, 1987; Psaltis, Park, & Hong, 1988):

$$y_i(x) = f \left[W_0(i) \sum_j^N W_i(i, j)x(j) + \sum_j^N \sum_k^M W_i(i, j, k)x(j)x(k) + \dots \right] \tag{2}$$

Figure 1. Higher-order neural network architecture-I

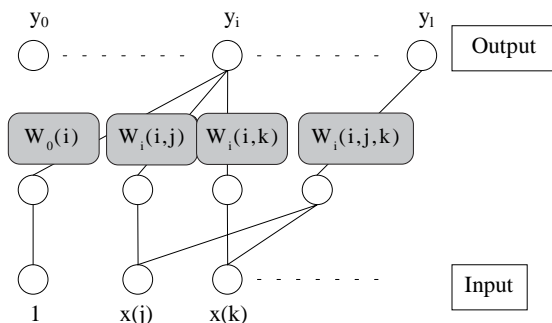
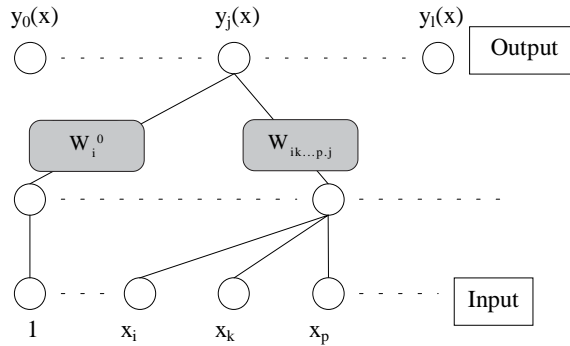


Figure 2. Higher-order neural network architecture-II



Neurons that include terms up to and including degree- k are referred to as k th-order neurons (nodes). Figure 1 further explains the subscripts i, j , and k used in Equation 2. The following alternative, simpler formulation is due to Lisboa and Perantonis (1991):

$$y_j(x) = f \left[W_i^0 + \sum_i \sum_k \dots \sum_p w_{ik...p,j} x_i \cdot x_k \dots x_p \right] \quad (3)$$

where a single weight is applied to all n -tuples $x_1 \dots x_p$ in order to generate output- y_i from that particular neuron.

This is reminiscent of Rumelhart, Hinton, and Williams (1986) formulation of their so-called “sigma-pi” neurons ($\sum w_{ij} \prod x_i x_j \dots x_k$), for which they show that the generalized Delta Rule (standard backpropagation) can be applied as readily as for simple additive neurons ($\sum w_{ij} x_i$). Moreover, the increased computational load resulting from the large increase in network weights means that the complex input-output mappings, normally only achievable in multilayered networks, can now be realized in a *single* HONN layer (Zhang & Fulcher, 2004).

In summary, HONN activation functions incorporate *multiplicative* terms.

Now the output of a k th-order single-layer HONN neuron will be a nonlinear function comprising polynomials of up to k th-order. Moreover, since no hidden layers are involved, both Hebbian and perceptron learning rules can be employed (Shin & Ghosh, 1991).

Multiplicative interconnections within ANNs have been applied to many different problems, including invariant pattern recognition (Giles, Griffin, & Maxwell, 1988; 1991; Goggin, Johnson, & Gustafson, 1993; Lisboa & Pentonis, 1991), however their complexity usually limits their usefulness.

Karayiannis and Venetsanopoulos (1993) make the observation that the performance of first-order ANNs can be improved, within bounds, by utilizing sophisticated learning algorithms. By contrast, HONNs can achieve superior performance *even if* the learning algorithm is based on the simpler outer-product rule.

A different approach was taken by Redding, Kowalczy, and Downs (1993) and involved the development of a constructive HONN architecture that solved the binary mapping in polynomial time. Central to this process was the selection of the multiplicative nonlinearities as hidden nodes within the HONN, depending on their relevance to the pattern data of interest.

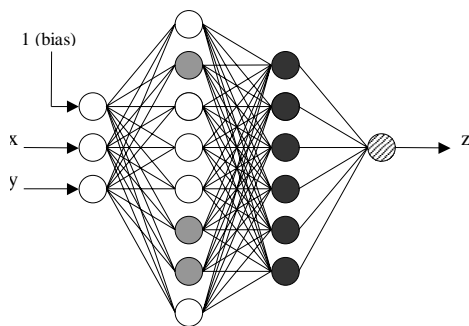
Polynomial HONNs

The authors have developed several different HONN models during the past decade or so. We now present a brief background on the development of polynomial, trigonometric, and similar HONN models. A more comprehensive coverage, including derivations of weight-update equations, is presented in Zhang and Fulcher (2004).

Firstly, all PHONNs described in this section utilize various combinations of linear, power, and multiplicative (and sometimes other) neuron types and are trained using standard backpropagation. The generic HONN architecture is shown in Figure 3, where there are two network inputs (independent variables) x and y , and a single network output (dependent variable) z .

In the first hidden layer, the white neurons are either $\cos(x)$ or $\sin(y)$, and the grey neurons either $\cos^2(x)$ or $\sin^2(y)$. All (black) neurons in the second hidden layer are multiplicative, and the (hashed) output neurons either linear (PHONN#1) or a sigmoid-logarithmic pair (PHONN#2), as described later. Some of the intermediate weights are fixed and some variable, according to the formulation of the polynomial being synthesized. All of the weights connecting the second hidden layer to the output layer are adaptable (PHONN#1,2). By contrast, only the latter are adjustable in PHONN#0; the first two-layer weights are fixed (=1).

Figure 3. Polynomial higher-order neural network (PHONN)



The first model (PHONN#0) facilitates extraction of the linear coefficients $a_{k_1k_2}$ from the general n th-order polynomial:

$$z_i(x, y) = \sum_{k_1k_2} a_{k_1k_2} x_i^{k_1} y_i^{k_2} \quad (4)$$

Now, since variable weights are present in only one layer here, PHONN#0 can be compared with Rosenblatt's (two-layer) perceptron, which is well-known to be limited to solving linearly separable problems.

In PHONN#1, the general n th-order polynomial of Equation 4 is expanded as follows:

$$z_i(x, y) = \sum_{k_1k_2=0}^n (a_{k_1k_2}^0) [a_{k_1k_2}^x x]^{k_1} [a_{k_1k_2}^y y]^{k_2} \quad (5)$$

Each coefficient from Equation 4 has now been replaced by three terms in Equation 5. Moreover, we now have *two* adjustable layers at our disposal, such that PHONN#1 has similar discrimination capability to an MLP.

The linear output neuron of PHONN#1 is replaced by a sigmoid-logarithmic neuron pair in PHONN#2, which leads to faster network convergence. Model PHONN#3 comprises groups of PHONN#2 neurons (ANN groups are discussed in the following section).

If we use a PHONN to simulate the training data, the model will "learn" the coefficients and order of the polynomial function. If we use adaptive HONN models to simulate the data, the models will *not only* "learn" the coefficients and order, *but also* the different

Table 1. \$A-\$US exchange rate (March 2005)

Date	Exchange Rate	Input#1 (X)	Input#2 (Y)	Desired Output (Z)
1	0.7847	0.093	0.000	0.057
2	0.7834	0.000	0.057	0.607
3	0.7842	0.057	0.607	0.650
4	0.7919	0.607	0.650	1.000
7	0.7925	0.650	1.000	0.686
8	0.7974	1.000	0.686	0.479
9	0.7930	0.686	0.479	0.729
10	0.7901	0.479	0.729	0.229
11	0.7936	0.729	0.229	0.429
14	0.7866	0.229	0.429	0.800
15	0.7894	0.429	0.800	0.714
16	0.7946	0.800	0.714	0.736
17	0.7935	0.714	0.736	
18	0.7937	0.736		

functions. In other words, the model “learns” the polynomial function if the data is in fact a polynomial function.

Now, instead of employing a polynomial series expansion, we could alternatively use a trigonometric one, as indicated in Equation 6.

$$Z = a_{00} + a_{01}\sin(y) + a_{02}\sin^2(y) + a_{10}\cos(x) + a_{11}\cos(x)\sin(y) + a_{12}\cos(x)\sin^2(y) + a_{20}\cos^2(y) + a_{21}\cos^2(x)\sin(y) + a_{22}\cos^2(x)\sin^2(y) + \dots \tag{6}$$

This naturally leads to the development of THONN models (and likewise THONN groups — see subsequent paragraphs).

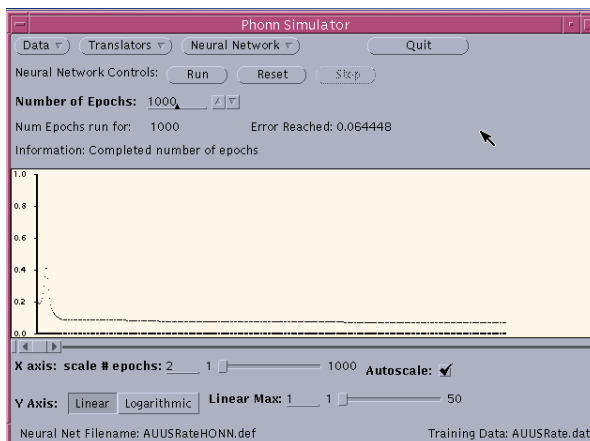
One significant feature of models P(T)HONN#1 and higher is that we have opened up the “black box” or closed architecture normally associated with ANNs. In other words, we are able to associate individual network weights with polynomial coefficients and vice versa. This is a significant finding, since users — especially those in the financial sector — invariably prefer explanations (justifications) for the decisions made by their predictors, regardless of the nature of the underlying decision engine.

We now proceed to illustrate the use of PHONNs by way of a simple example, namely exchange-rate prediction. Table 1 shows how the Australian-US dollar exchange rate varied during March 2005 (Federal Reserve Board, 2005).

The following formula was used to scale the data to within the range 0 to 1, in order to meet constraints:

$$\frac{\{(individual_rate) - (lowest_rate)\}}{\{(highest_rate) - (lowest_rate)\}} \tag{7}$$

Figure 4. PHONN Simulator main window



Equation 7 was applied to each separate entry of a given set of simulation data — in other words, the *individual_rate*. The smallest entry in the data set serves as the *lowest_rate*, and the largest entry as the *highest_rate*. After applying Equation 7, the data have been converted into the Input#1 column of Table 1.

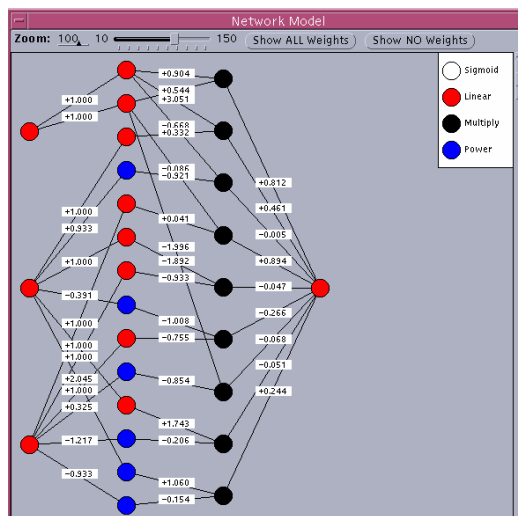
We use previous day and current-day exchange rates to predict the next day's rate. Accordingly, we copy the data from 3/2/2005 to 3/18/2005 to the Input#2 column of Table 1 — this being the second input to the PHONN — and copy the data from 3/3/2005 to 3/18/2005 to the Output column of Table 1 (in other words, the desired PHONN output).

The PHONN simulation system was written in the C language, and runs under X-Windows on a SUN workstation. It incorporates a user-friendly graphical user interface (GUI), which enables any step, data, or calculation to be reviewed and modified dynamically in different windows. At the top of the PHONN simulator main window are three pull-down menus: Data, Translators, and Neural Network, as illustrated in Figure 4 (which, by the way, shows the result of network training using the data of Table 1).

Each of these offers several options, and selecting a particular option creates another window for further processing. For instance, once we have selected a data set via the Data menu, two options are presented for data loading and graphical display.

Data is automatically loaded when the Load option is selected. Alternatively, the Display option displays data not only in graphical form, but also translated, if so desired (e.g., rotation, elevation, grids, smooth, influence, etc.). The Translators menu is used to convert the selected raw data into network form, while the Neural Network menu is used to convert the data into a nominated model (an example of which appears in Figure 5). These two menus allow the user to select different models and data, in order to generate

Figure 5. “PHONN Network Model” subwindow



and compare results. Figure 5 shows the network weights resulting from training using the data of Table 1.

All of the previous steps can be simply performed using a mouse. Hence, changing data or network model and comparing results can all be achieved easily and efficiently.

There are more than twelve windows and subwindows in the PHONN Simulator system; both the system mode and its operation can be viewed dynamically, in terms of:

- Input/output data,
- Neural network models,
- Coefficients/parameters, and so on.

Figure 6. “Load Network Model File” subwindow

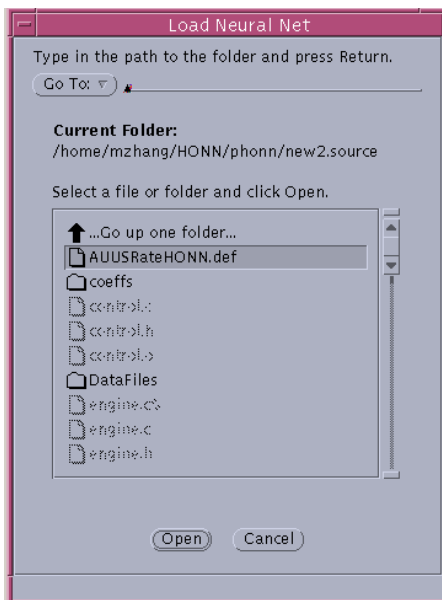
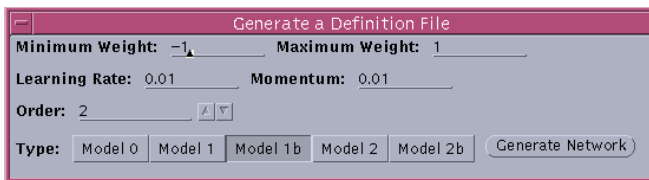


Figure 7. “Generate Definition File” subwindow

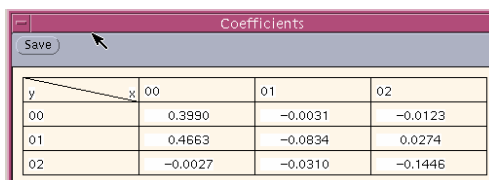


The simulator operates as a general neural network system and includes the following functions:

- Load a data file,
- Load a neural network model (Figure 6) for Table 1 data,
- Generate a definition file (Figure 7) for Table 1 data,
- Write a definition file,
- Save report,
- Save coefficients (Figure 8) for Table 1 data, and so on.

The “System mode” windows allow the user to view, in real time, how the neural network model learns from the input training data (in other words, how it extracts the weight values).

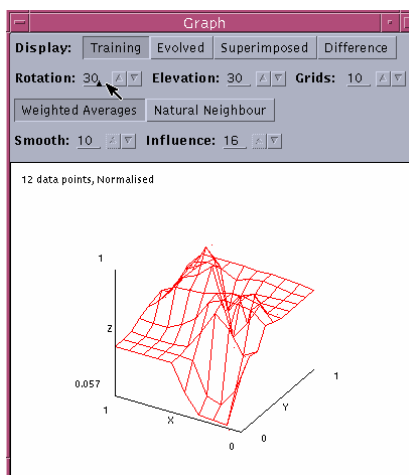
Figure 8. “Coefficients” subwindow



The screenshot shows a window titled "Coefficients" with a "Save" button. Below the button is a table with the following data:

y \ x	00	01	02
00	0.3990	-0.0031	-0.0123
01	0.4663	-0.0834	0.0274
02	-0.0027	-0.0310	-0.1446

Figure 9. “Graph” subwindow



When the system is running, the following system-mode windows can be opened simultaneously from within the main window:

- “Display Data (Training, Evolved, Superimposed, and Difference),”
- “Show/Set Parameters,”
- “Network Model (including all weights),” and
- “Coefficients.”

Thus, every aspect of the system’s operation can be viewed graphically.

A particularly useful feature of this system is that one is able to view the mode, modify it, or alternatively change other parameters *in real time*. For example, when the user chooses the “Display Data” window to view the input-training data file, they can change the graph format for the most appropriate type of display (in other words, modify the graph’s rotation, elevation, grids, smoothing, and influence).

During data processing, the “Display Data” window offers four different models to display the results, which can be changed in real time, namely: “Training,” “Evolution,” “Superimposed,” and “Difference (using the same format selected for the input data),” as indicated in Figure 9 for Table 1 data.

- “Training” displays the data set used to train the network,
- “Evolved” displays the data set produced by the network (and is unavailable if a network definition file has not been loaded),
- “Superimposed” displays both the training and the evolved data sets together (so they can be directly compared in the one graph), and
- “Difference” displays the difference between the “Training” and the “Evolved” data sets.

Figure 10. Report generation within PHONN Simulator

```
***** NETWORK PERFORMANCE FIGURES *****
Input-0  Input-1  Act0/P-0  Des0/P-0  Error  %Error
0.093000  0.000000  0.423025  0.057000  0.366025  36.80
0.000000  0.057000  0.411523  0.607000  -0.195477  19.55
0.057000  0.607000  0.619724  0.650000  -0.030276  3.03
0.607000  0.650000  0.648283  1.000000  -0.351717  35.17
0.650000  1.000000  0.748960  0.686000  0.062960  6.30
1.000000  0.686000  0.557296  0.479000  0.078296  7.83
0.686000  0.479000  0.586336  0.729000  -0.142664  14.27
0.479000  0.729000  0.684657  0.229000  0.455657  45.57
0.729000  0.229000  0.511255  0.429000  0.082255  8.23
0.229000  0.429000  0.589772  0.800000  -0.210228  21.02
0.429000  0.800000  0.708355  0.714000  -0.005645  0.56
0.800000  0.714000  0.629063  0.736000  -0.106931  10.63

Ave Error Per Pattern = 0.174011
Ave Percent Error Per Pattern = 17.4011%
```

The “Rotation” command changes the angle of rotation at which the wire-frame mesh is projected onto the screen. This allows the user to “fly” around the wire-frame surface. The default value is 30°, but is adjustable from 0° to 355° in increments of 5°, with wraparound from 360° to 0° (this value can be simply adjusted with either the “up/down” buttons or by entering a number directly).

“Elevation” changes the angle of elevation at which the wire-frame mesh is projected onto the screen. This allows the user to “fly” either above or below the wire-frame surface (usage is similar to Rotation).

The “Grids” command changes the number of wires used in the wire-frame mesh. It is adjustable between 6 and 30, using either the “up/down” buttons or by directly entering a number. Low grid numbers allow fast display, but with decreased resolution; high numbers provide a more accurate rendition of the surface, but at the cost of increased display time.

If the user is not satisfied with the results and wants a better outcome (a higher degree of model accuracy), they can stop the processing and set *new* values for the model parameters, such as learning rate, momentum, error threshold, and random seed. The neural network model can be easily changed as well.

As usual with neural network software, the operating procedure is as follows:

Step 1: Data pre-processing (encoding),

Step 2: Load and view data,

Step 3: Choose and load neural network model,

Step 4: Show/Set the network parameters,

Step 5: Run the program,

Step 6: Check the results:

If satisfactory, then go to Step 7, otherwise go to Step 3,

Step 7: Save and export the results,

Step 8: Data decoding (postprocessing).

There are two basic requirements that must be satisfied before the PHONN simulator is able to start running: One is *input training data* and the other is *input the network*. The users must also have *loaded some training data* and *loaded a network*.

Figure 10 shows the running report for Table 1 data, from which we see the average error is 17.4011%. Returning to Figure 8, we see that the following formula can be used to represent the data of interest (exchange rate), thereby relating network weights to polynomial coefficients:

$$Z=0.3990-0.0031X-0.0123X*X+0.4663Y-0.0834X*Y-0.0274X*X*Y-0.0027Y*Y-0.0310X*Y*Y-0.1446X*X*Y*Y \quad (8)$$

HONN Groups

Prior to our development of ANN groups, we were aware of earlier work on groups of individual neurons (Hu & Pan, 1992; Willcox, 1991). What motivated our development of firstly ANN groups and thenceforth P(T)HONN groups was the poor performance of ANNs on human-face recognition, which we investigated in the context of airport security (Zhang & Fulcher, 1996).

It is possible to define a neural network group in the usual set theory terms, as follows:

$$ANN = MLP \cup SOM \cup RBF \cup ART \cup SVM \dots \quad (9)$$

MLP is thus a subset of the set *ANN*; likewise a particular instance of *MLP* (say *MLP*_{100:70:20}) is a subset of *MLP*. Moreover, providing either the sum and/or product can be defined for every two elements in a nonempty set $N \subset ANN$ (Inui, Tanabe & Onodera, 1978; Naimark & Stern, 1982), and then we refer to this set as a neural network group.

ANN groups are particularly useful in situations involving discontinuous data, since we can define piecewise function groups, as follows:

$$\begin{aligned} O_i + O_j &= O_i && (A < I < B) \\ &= O_j && (B < I < C) \end{aligned} \quad (10)$$

where I = ANN input, O = ANN output, and for every two elements $n_i, n_j \in N$, the sum $n_i + n_j$ is a piecewise function.

Now in the same vein as Hornik (1991) and Leshno (1993), it is possible to show that piecewise function groups (of MLPs employing locally bounded, piecewise continuous activation functions and thresholds) are capable of approximating any piecewise continuous function, to any degree of accuracy (Zhang, Fulcher, & Scofield, 1997).

Not surprisingly, such ANN groups offer superior performance compared with ANNs when dealing with discontinuous, nonsmooth, complex training data, which is often the case with financial time series.

HONN Applications

The three application areas in which we have focused our endeavors to date are (1) human-face recognition (Zhang & Fulcher, 1996), (2) satellite weather forecasting (Zhang, Fulcher, & Scofield, 1997; Zhang & Fulcher, 2004), and (3) financial time-series prediction (Zhang, Xu, & Fulcher, 2002; Zhang, Zhang, & Fulcher, 2000). In each case, we are typically dealing with discontinuous, nonsmooth, complex training data, and thus HONN (and HONN groups) come into their own.

Automatic Face Recognition

Automatic (1-of- n) facial recognition is a complex pattern recognition task, especially for real-time operation under variable lighting conditions, where faces are tilted or rotated, and where n is large. There can also be significant repercussions for false positives, and more especially false negatives (that is, failure to detect a “wanted person”). The advantage of using ANN groups in such an application is that if one particular ANN model cannot perform the desired recognition, then perhaps another model belonging to the ANN set (group) can do better.

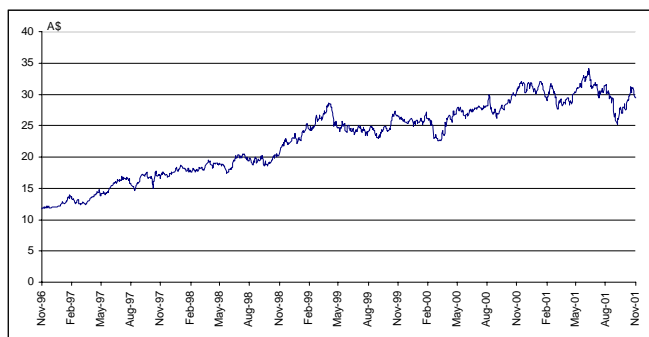
Using ANN group trees can extend this approach further. Nodes and interconnecting weights in such trees grow adaptively during the training process, according to both the desired number of “wanted” leaf-node faces and the variability contained within the training exemplars. As a result, such a network is capable of recognizing tilted or rotated facial images as being the same person; in other words, it can handle topological deformations and/or 3D translations. Zhang and Fulcher (1996) describe such ANN group trees in terms of Tolerance Space Theory (Chen, 1981; Zeeman, 1962).

Group-based adaptive-tolerance (GAT) trees have been successfully applied to automatic face recognition (Zhang & Fulcher, 1996). For this study, ten (28*28 pixel, 256-level gray scale) images of 78 different faces (front, tilted, rotated, smiling, glasses, beard, etc.) were used for both training (87) and testing (693) purposes. For front-face recognition, the error rate was 0.15% (1 face); for tilted and rotated faces (of up to 15%), the error rates were 0.16% and 0.31%, respectively. Thus GAT trees were more “tolerant” in their classification.

Rainfall Estimation

Global weather prediction is acknowledged as one of computing’s “grand challenges” (Computing Research Associates, 2005). The world’s fastest (parallel vector)

Figure 11. Commonwealth Bank of Australia share prices (November 1996-November 2001)



supercomputer — at least up until 2004 (Meuer, Stronmaier, Dongarra, & Simon, 2005) — was devoted to simulation of the earth for purposes of global weather forecasting. Rainfall estimation is a complicated, nonlinear, discontinuous process. Single ANNs are unable to deal with discontinuous, nonsmooth input training data; ANN groups, on the other hand, are well-suited to such problems.

ANNs and ANN groups both outperform conventional rainfall estimation, yielding error rates of around 17% and 3.9%, respectively (compared with ~30.4% with the latter) (Zhang & Fulcher, 2004; Zhang, Fulcher, & Scofield, 1997). ANN groups were subsequently used as the reasoning engine within the ANSER Expert System developed for satellite-derived rainfall estimation.

In Zhang and Fulcher (2004), PHONN variants (PT-, A- and M-) are applied to half-hourly rainfall prediction. Another model — the Neuron-Adaptive HONN (described in the next section) — led to a marginal error reduction (3.75%).

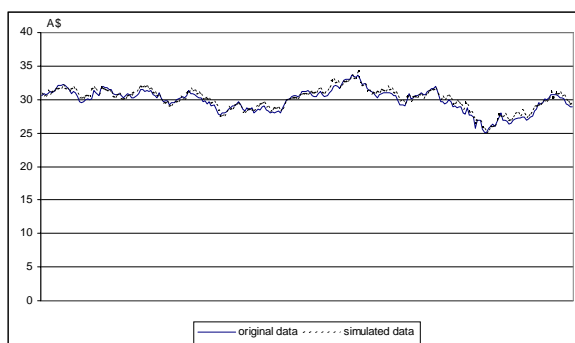
Another variant — the Sigmoid PHONN — has been shown to offer marginal performance improvement over both PHONN and M-PHONN when applied to rainfall estimation (more specifically, 5.263% average error compared with 6.36% and 5.42%, respectively) (Zhang, Crane, & Bailey, 2003).

Application of HONNs to Financial Time Series Data

Both polynomial and trigonometric HONNs have been used to both simulate and predict financial time-series data (Reserve Bank of Australia Bulletin, 2005), to around 90% accuracy (Zhang, Murugesan, & Sadeghi, 1995; Zhang, Zhang, & Keen, 1999).

In the former study, all the available data was used during training. In the latter, the data was split in two — one half being used for training and the other half for testing (and where data from the previous 2 months was used to predict the next month's data). More

Figure 12. Simulation of Commonwealth Bank of Australia share prices (November 2000-November 2001) using NAHONN



recently, the polynomial/trigonometric HONN (Lu & Zhang, 2000) and Multiple-PHONN (Zhang & Lu, 2001) models have been shown to offer improved performance, compared with P(T)HONNs.

It was mentioned previously that PHONN Model#3 comprises groups of PHONN#2 neurons. When applied to financial time-series prediction, PHONN groups produce up to an order of magnitude performance improvement over PHONNs — more specifically around 1.2% error for simulation (compared with 11%) and 5.6% error for prediction (compared with 12.7%) (Zhang, Zhang, & Fulcher, 2000). Similar improvements in performance are observed with THONN groups (Zhang, Zhang, & Fulcher, 1996, 1997).

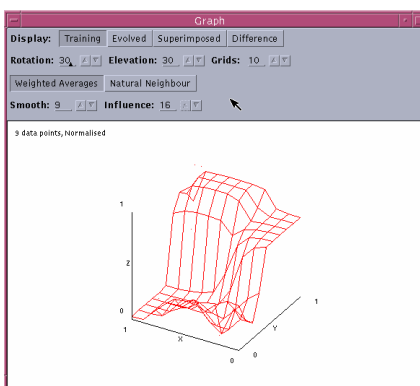
The neuron-adaptive HONN (and NAHONN group) leads to faster convergence, much reduced network size and more accurate curve fitting, compared with P(T)HONNs (Zhang, Xu, & Fulcher, 2002). Each element of the NAHONN group is a standard multilayer HONN comprising adaptive neurons, but which employs locally bounded, piecewise continuous (rather than polynomial) activation functions and thresholds.

The (1-Dimensional) neuron activation function is defined as follows:

Table 2. \$A-\$US exchange rate (2004)

Australian Dollar/U.S. Dollar Exchange Rate (2004)				
Month	Raw Exchange Rate	Input 1#1	Input#2	Desired Output
January	0.7576	0.95	0.93	1.00
February	0.7566	0.93	1.00	0.96
March	0.761	1.00	0.96	0.37
April	0.7584	0.96	0.37	0.03
May	0.7198	0.37	0.03	0.26
June	0.697	0.03	0.26	0.14
July	0.7125	0.26	0.14	0.00
August	0.7042	0.14	0.00	0.43
September	0.6952	0.00	0.43	0.75
October	0.7232	0.43	0.75	
November	0.7447	0.75		

Figure 13. Input data for Sigmoid PHONN simulation (prediction)



$$\psi_{i,k}(net_{i,k}) = o_{i,k}(net_{i,k}) = \sum_{h=1}^s f_{i,k,h}(net_{i,k}) \tag{11}$$

where $net_{i,k}$ is the input (internal state) of the i th neuron in the k th layer, and $w_{i,j,k}$ is the weight connecting the j th neuron in layer-($k-1$) with the i th neuron in layer- k . This formulation, along with nD and *multi* n-Dimensional NAHONNs, incorporate free parameters which can be adjusted, along with the weights, during training (unlike conventional feed-forward ANNs). The NAHONN learning algorithm is based on steepest descent, but since the hidden-layer variables are adjustable, NAHONN offers more flexibility and more accurate approximation capability compared with (fixed activation function) MLP/BPs (Zhang, Xu, & Fulcher, 2002).

In one comparative experiment, a NAHONN with nonlinear neuron activation function led to around half the RMS error compared with PHONN, and a NAHONN which utilized piecewise NAFs required less than half the number of hidden-layer neurons, converged in less than a third of the time and led to an RMS output error two orders of magnitude lower than PHONN (Zhang, Xu, & Fulcher, 2002; Figure 12).

Now as with the earlier P(T)HONN groups, it is possible to prove a similar general result to that found previously by Hornik (1991) for ANNs, namely that NAHONN groups are capable of approximating any kind of piecewise-continuous function to any degree of accuracy (a proof is provided in Zhang, Xu, & Fulcher, 2002). Moreover, these models are capable of automatically selecting not only the optimum model for a particular time series, but also the appropriate model order.

Returning to the Sigmoid PHONN (Zhang, Crane, & Bailey, 2003), the \$Australian-\$US exchange rate data of Table 2 was used to predict the following month's rate — in other words, based on the previous two months rates, as follows:

$$input\#1 = \frac{current_month - \min imum}{\max imum - \min imum} \tag{12}$$

Figure 14. Sigmoid PHONN training (convergence)

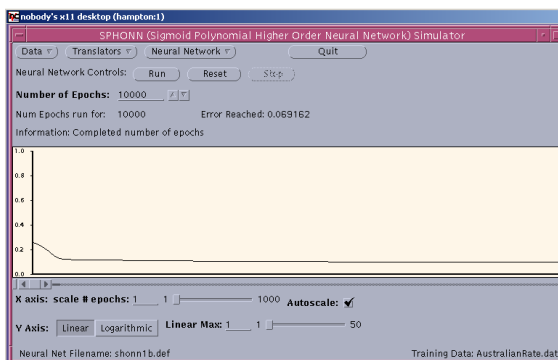


Figure 15. Sigmoid PHONN network weights

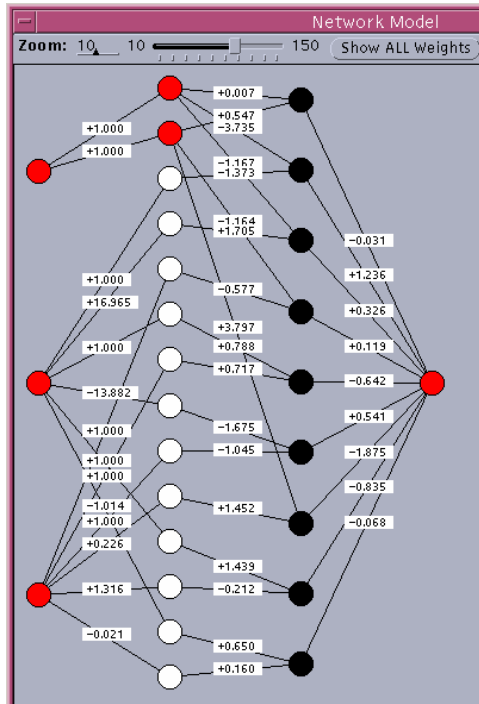
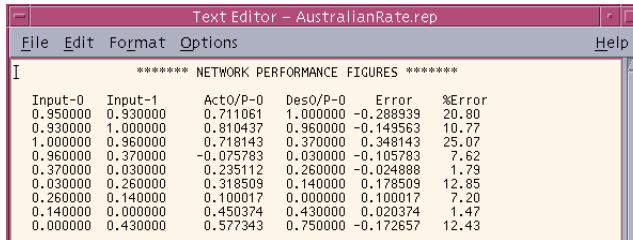


Figure 16. Sigmoid PHONN network performance



$$input\#2 = \frac{next_month - \min imum}{\max imum - \min imum} \tag{13}$$

$$desired_output = \frac{month_after_next - \min imum}{\max imum - \min imum} \tag{14}$$

Figure 17. HONN performance comparison (Reserve Bank of Australia: Credit-card lending, August 1996-June 1997)

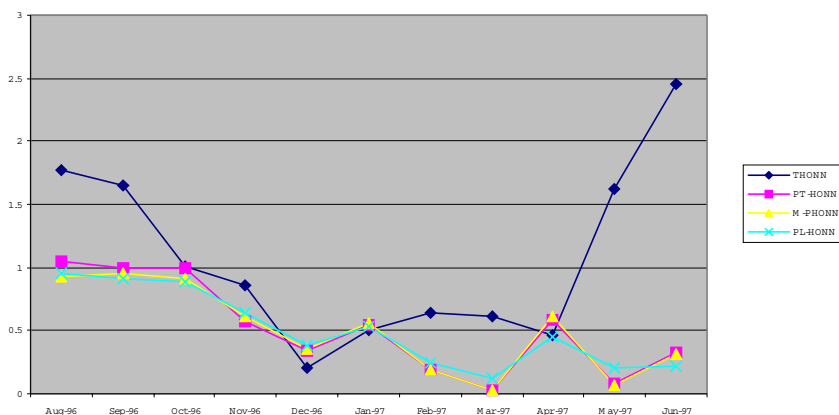
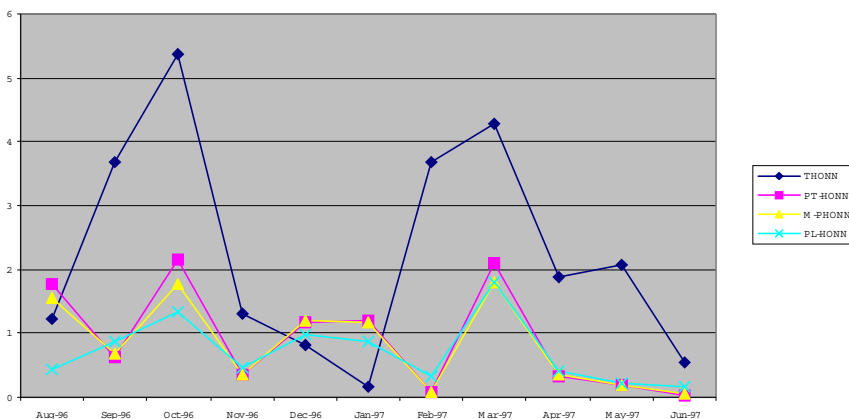


Figure 18. HONN performance comparison (Reserve Bank of Australia: Dollar-yen exchange rate (August 1996-June 1997)



Thus, only the data within the box was used for this exercise.

This input training data is plotted in Figure 13 (X = input#1, Y = input#2, and Z = desired output, respectively).

Convergence of the Sigmoid PHONN is shown in Figure 14 and the final network weights in Figure 15. In this example, convergence occurs after roughly 700 epochs, despite Figure 14 showing 10,000 epochs total (and to an error of around 7%).

In Figure 15, the third (uppermost, lefthand side) network input is the bias term, the remaining ones being input#1 (based on the current monthly rate) and input#2 (based

on the next month's exchange rate); the network output is the desired (month after next) currency exchange rate, as previously explained. The grey neurons are sigmoid types, the white neurons linear, and the black ones multiplicative.

The performance of this Sigmoid PHONN (actual vs. desired outputs) is summarized in Figure 16.

More recently, Zhang (2003) has developed a *multi*PHONN, which employs a logarithmic activation function, and as its name suggests, is capable of simulating not only polynomial and/or trigonometric functions, but also combinations of these, as well as sigmoid and/or logarithmic functions. As a result, they are better able to approximate real world economic time series data. It can be seen in Figures 17 and 18 that PL-HONN offers significant performance improvement over THONNs, and marginal improvement over both PT- and M-PHONNs, when applied to typical financial time series data (Reserve Bank of Australia Bulletin: www.abs.gov.au/ausstats/abs@.nsf/w2.3).

The main finding from these experiments is that the more sophisticated PHONN variants significantly outperform THONN on typical financial time-series data, however all yield significantly lower errors compared with conventional feed-forward ANNs (not shown in this chapter's figures).

Conclusion

We have introduced the concepts of higher-order artificial neural networks and ANN groups. Such models offer significant advantages over classical feed-forward ANN models such as MLP/BP, due to their ability to better approximate complex, nonsmooth, often discontinuous training data. Important findings about the general approximation ability of such HONNs (and HONN groups) have been presented, which extend the earlier findings of Hecht-Nielsen (1987), Hornik (1991), and Leshno, Lin, Pinkus, and Schoken (1993).

Acknowledgments

We would like to acknowledge the financial assistance of the following organizations in our development of higher-order neural networks: Societe Internationale de Telecommunications Aeronautique, Fujitsu Research Laboratories, Japan, the U.S. National Research Council, and the Australian Research Council.

References

- Allen, D. W., & Taylor, J. G. (1994). Learning time series by neural networks. In M. Marinaro & P. Morasso (Eds.), *Proceedings of the International Conference on Neural Networks, Sorrento, Italy* (pp. 529-532). Berlin: Springer.
- Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *J. Financial Economics*, 51(2), 245-271.
- Azema-Barac, M. E., & Refenes, A. N. (1997). Neural networks for financial applications. In E. Fiesler, & R. Beale (Eds.), *Handbook of neural computation* (G6.3; pp. 1-7). Oxford, UK: Oxford University Press.
- Azoff, E. (1994). *Neural network time series forecasting of financial markets*. New York: Wiley.
- Back, A. (2004). Independent component analysis. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 59-95). Berlin: Springer.
- Barron, R., Gilstrap, L., & Shrier, S. (1987). Polynomial and neural networks: Analogies and engineering applications. In *Proceedings of the International Conference on Neural Networks, New York, Vol. 2* (pp. 431-439).
- Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis: Forecasting and control* (rev. ed.). San Francisco: Holden Day.
- Brockwell, P. J., & Davis, R. A. (1991). *Time series: Theory and methods* (2nd ed.). New York: Springer.
- Campolucci, P., Capparelli, F., Guarnieri, S., Piazza, F., & Uncini, A. (1996). Neural networks with adaptive spline activation function. In *Proceedings of the IEEE MELECON'96 Conference, Bari, Italy* (pp. 1442-1445).
- Caudill, M., & Butler, C. (1990). *Naturally intelligent systems*. Cambridge, MA: MIT Press.
- Chakraborty, K., Mehrotra, K., Mohan, C., & Ranka, S. (1992). Forecasting the behaviour of multivariate time series using neural networks. *Neural Networks*, 5, 961-970.
- Chang, P. T. (1997). Fuzzy seasonality forecasting. *Fuzzy Sets and Systems*, 112, 381-394.
- Chatfield, C. (1996). *The analysis of time series: An introduction*. London: Chapman & Hall.
- Chen, C. T., & Chang, W. D. (1996). A feedforward neural network with function shape autotuning. *Neural Networks*, 9(4), 627-641.
- Chen, L. (1981). Topological structure in visual perception. *Science*, 218, 699.
- Chen, S.-H. (Ed.). (2002). *Genetic algorithms and genetic programming in computational finance*. Boston: Kluwer.
- Computing Research Associates, (2005). Retrieved April 2005, from www.cra.org
- Devroye, L., Györfi, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. New York: Springer.

- Diamond, I., & Jeffries, J. (2001). *Beginning statistics: An introduction for social sciences*. London: Sage Publications.
- Edelman, D., & Davy, P. (2004). Adaptive technical analysis in the financial markets using machine learning: A statistical view. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 1-16). Berlin: Springer.
- Federal Reserve Board. (2005). Retrieved April 2005, from <http://www.federalreserve.gov/releases/>
- Giles, L., Griffin, R., & Maxwell, T. (1988). Encoding geometric invariances in high-order neural networks. In D. Anderson (Ed.), *Proceedings Neural Information Processing Systems* (pp. 301-309).
- Giles, L., & Maxwell, T. (1987). Learning, invariance and generalisation in high-order neural networks. *Applied Optics*, 26(23), 4972-4978.
- Goggin, S., Johnson, K., & Gustafson, K. (1993). A second-order translation, rotation and scale invariant neural network. In R. Lippmann, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in neural information processing systems 3*. San Mateo, CA: Morgan Kaufman.
- Gorr, W. L. (1994). Research perspective on neural network forecasting. *International Journal of Forecasting*, 10(1), 1-4.
- Harris, R., & Sollis, R. (2003). *Applied time series modelling and forecasting*. Chichester, UK: Wiley.
- Harvey, A. C. (1989). *Forecasting, structural times series models and the kalman filter*. Cambridge, UK: Cambridge University Press.
- Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the International Conference on Neural Networks, Vol. 3* (pp. 11-13). New York: IEEE Press.
- Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4, 251-257.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multi-layer feedforward networks are universal approximators. *Neural Networks*, 2, 359-366.
- Hu, S., & Yan, P. (1992). Level-by-level learning for artificial neural groups. *Electronica Sinica*, 20, 10, 39-43.
- Hu, Z., & Shao, H. (1992). The study of neural network adaptive control systems. *Control and Decision*, 7, 361-366.
- Iba, H., & Sasaki, T. (1999). Using genetic programming to predict financial data. In *Proceedings of the 1999 Congress on Evolutionary Computation — CEC99, Vol. 1* (pp. 244-251). New Jersey: IEEE Press.
- Inui, T., Tanabe, Y., & Onodera, Y. (1978). *Group theory and its application in physics*. Berlin: Springer.
- Karayiannis, N., & Venetsanopoulos, A. (1993). *Artificial neural networks: Learning algorithms, performance evaluation and applications*. Boston: Kluwer.

- Kingdon, J. (1997). *Intelligent systems and financial forecasting*. Berlin: Springer.
- Kosmatopoulos, E. B., Polycarpou, M. M., Christodoulou, M. A., & Ioannou, P. A. (1995). High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, 6(2), 422-431.
- Lapedes, A. S., & Farber, R. (1987). Non-linear signal processing using neural networks: Prediction and system modelling. *Los Alamos National Laboratory* (Technical Report LA-UR-87).
- Lee, Y. C., Doolen, G., Chen, H., Sun, G., Maxwell, T., Lee, H., et al. (1986). Machine learning using a higher order correlation network. *Physica D: Nonlinear Phenomena*, 22, 276-306.
- Leshno, M., Lin, V., Pinkus, A., & Schoken, S. (1993). Multi-layer feedforward networks with a non-polynomial activation can approximate any function. *Neural Networks*, 6, 861-867.
- Lippmann, R. P. (1989). Pattern classification using neural networks. *IEEE Communications Magazine*, 27, 47-64.
- Lisboa, P., & Perantonis, S. (1991, November 18-21). Invariant pattern recognition using third-order networks and zernlike moments. In *Proceedings of the IEEE International Joint Conference on Neural Networks, Singapore, Vol. II* (pp. 1421-1425).
- Lu, B., & Zhang, M. (2000, May 15-17). Using PT-HONN models for multi-polynomial function simulation. In *Proceedings of IASTED International Conference on Neural Networks*, Pittsburgh, PA.
- Makridakis, S., Andersoan, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., et al. (1982). The accuracy of extrapolation methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2), 111-153.
- Meuer, H., Stronmaier, E., Dongarra, J., & Simon, H. D. (2005). Retrieved April 2005, from <http://www.top500.org>
- Mills, T. C. (1993). *The econometric modelling of financial time series*. Cambridge, UK: Cambridge University Press.
- Naimark, M., & Stern, A. (1982). *Theory of group representation*. Berlin: Springer.
- Plotkin, H. (1993). *Darwin machines and the nature of knowledge*. Cambridge, MA: Harvard University Press.
- Psaltis, D., Park, C., & Hong, J. (1988). Higher order associative memories and their optical implementations. *Neural Networks*, 1, 149-163.
- Redding, N., Kowalczyk, A., & Downs, T. (1993). Constructive higher-order network algorithm that is polynomial time. *Neural Networks*, 6, 997-1010.
- Refenes, A. N. (Ed.). (1994). *Neural networks in the capital markets*. Chichester, UK: Wiley.
- Reid, M. B., Spirkovska, L., & Ochoa, E. (1989). Simultaneous position, scale, rotation invariant pattern classification using third-order neural networks. *International Journal of Neural Networks*, 1, 154-159.

- Reinsel, G. C. (1997). *Elements of multivariate time series analysis*. New York: Springer.
- Reserve Bank of Australia Bulletin. (2005). Retrieved April 2005, from <http://abs.gov.au/ausstats/>
- Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. In D. Rumelhart, & J. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1 — Foundations*. Cambridge, MA: MIT Press.
- Schoneburg, E. (1990). Stock market prediction using neural networks: A project report. *Neurocomputing*, 2, 17-27.
- Shin, Y., & Ghosh, J. (1991, July). The Pi-Sigma Network: An efficient higher-order neural network for pattern classification and function approximation. In *Proceedings of the International Joint Conference on Neural Networks*, Seattle, WA (pp. I: 13-18).
- Sisman-Yilmaz, N. A., Alpaslan, F. N., & Jain, L. C. (2004). Fuzzy mulivariate auto-regression method and its application. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 281-300). Berlin: Springer.
- Tay, E. H., & Cao, L. J. (2001). Application of support vector machines in financial time series forecasting. *Omega*, 29, 309-317.
- Thimm, G., & Fiesler, E. (1997). High-order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, 8(2), 349-359.
- Trippi, R. R., & Turban, E. (Eds.). (1993). *Neural networks in finance and investing*. Chicago: Probus.
- Tseng, F. M., Tzeng, G. H., Yu, H. C., & Yuan, B. J. C. (2001). Fuzzy ARIMA model for forecasting the foreign exchange market. *Fuzzy Sets and Systems*, 118, 9-19.
- University of California, Irvine. (2005). Retrieved April 2005, from <http://www.ics.uci.edu/~mlern/MLrepository.html>
- Vecci, L., Piazza, F., & Uncini, A. (1998). Learning and approximation capabilities of adaptive spline activation function neural networks. *Neural Networks*, 11, 259-270.
- Vemuri, V., & Rogers, R. (1994). *Artificial neural networks: Forecasting time series*. Piscataway, NJ: IEEE Computer Society Press.
- Watada, J. (1992). Fuzzy time series analysis and forecasting of sales volume. In J. Kacprzyk, & M. Fedrizzi (Eds.), *Studies in fuzziness Vol.1: Fuzzy regression analysis*. Berlin: Springer.
- Weigend, A. S., & Gershenfeld, N. A. (Eds.). (1993). *Time series prediction: Forecasting the future and understanding the past*. Reading, MA: Addison Wesley.
- Welstead, S. T. (1994). *Neural networks and fuzzy logic applications in C/C++*. New York: Wiley.
- Wilcox, C. (1991). Understanding hierarchical neural network behaviour: A renormalization group approach. *Journal of Physics A*, 24, 2644-2655.
- Wolpert, D. H. (1996a). The existence of a priori distinctions between learning algorithms. *Neural Computation*, 8, 1391-1420.

- Wolpert, D. H. (1996b). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8, 1341-1390.
- Wood, J., & Shawe-Taylor, J. (1996). A unifying framework for invariant pattern recognition. *Pattern Recognition Letters*, 17, 1415-1422.
- Yamada, T., & Yabuta, T. (1992). Remarks on a neural network controller which uses an auto-tuning method for nonlinear functions. In *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2 (pp. 775-780).
- Zeeman, E. (1962). The topology of the brain and visual perception. In M. Fork, Jr. (Ed.), *Topology of 3-manifolds and related topics*. Englewood Cliffs, NJ: Prentice Hall.
- Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14, 35-62.
- Zhang, M. (2003, May 13-15). Financial data simulation using PL-HONN Model. In *Proceedings IASTED International Conference on Modelling and Simulation*, Marina del Rey, CA (pp. 229-233).
- Zhang, M., Crane, J., & Bailey, J. (2003, June 21-24). Rainfall estimation using SPHONN model. In *Proceedings of the International Conference on Artificial Intelligence*, Las Vegas, NV (pp. 695-701).
- Zhang, M., & Fulcher, J. (1996). Face recognition using artificial neural network group-based adaptive tolerance (GAT) trees. *IEEE Transactions on Neural Networks*, 7(3), 555-567.
- Zhang, M., & Fulcher, J. (2004). Higher order neural networks for satellite weather prediction. In J. Fulcher, & L. C. Jain (Eds.), *Applied intelligent systems: New directions* (pp. 17-57). Berlin: Springer.
- Zhang, M., Fulcher, J., & Scofield, R. (1997). Rainfall estimation using artificial neural network group. *Neurocomputing*, 16(2), 97-115.
- Zhang, M., & Lu, B. (2001, July). Financial data simulation using M-PHONN model. In *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC (pp. 1828-1832).
- Zhang, M., Murugesan, S., & Sadeghi, M. (1995, October 30-November 3). Polynomial higher order neural network for economic data simulation. In *Proceedings of the International Conference on Neural Information Processing*, Beijing, China (pp. 493-496).
- Zhang, M., Zhang, J. C., & Fulcher, J. (1996, March 23-25). Financial simulation system using higher order trigonometric polynomial neural network group model. In *Proceedings of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering*, New York, NY (pp. 189-194).
- Zhang, M., Zhang, J. C., & Fulcher, J. (1997, June 8-12). Financial prediction system using higher order trigonometric polynomial neural network group model. In *Proceedings of the IEEE International Conference on Neural Networks*, Houston, TX (pp. 2231-2234).
- Zhang, M., Zhang, J. C., & Fulcher, J. (2000). Higher-order neural network group models for data approximation. *International Journal of Neural Systems*, 10(2), 123-142.

Zhang, M., Zhang, J. C., & Keen, S. (1999, August 9-12). Using THONN system for higher frequency non-linear data simulation and prediction. In *Proceedings of the IASTED International Conference on Artificial Intelligence & Soft Computing*, Honolulu, HI (pp. 320-323).

Zhang, M., Xu, S., & Fulcher, J. (2002). Neuron-adaptive higher-order neural network models for automated financial data modelling. *IEEE Transactions on Neural Networks*, 13(1), 188-204.

Additional Sources

Machine Learning Databases: <http://www.ics.uci.edu/~mlearn/MLrepository.html>

Australian Bureau of Statistics, 19. Free sample data: <http://www.abs.gov.au/ausstats/abs@.nsf/w2.3>

National Library of Australia. Financial indicators: <http://www.nla.gov.au/oz/stats.html>

Chapter VI

Hierarchical Neural Networks for Modelling Adaptive Financial Systems

Masoud Mohammadian, University of Canberra, Australia

Mark Kingham, University of Canberra, Australia

Abstract

In this chapter, an intelligent hierarchical neural network system for prediction and modelling of interest rates in Australia is developed. A hierarchical neural network system is developed to model and predict 3 months' (quarterly) interest-rate fluctuations. The system is further trained to model and predict interest rates for 6-month and 1-year periods. The proposed system is developed with first four and then five hierarchical neural networks to model and predict interest rates. Conclusions on the accuracy of prediction using hierarchical neural networks are also reported.

Introduction

The prediction of uncertain dynamic systems, which are subject to external disturbances, uncertainty, and sheer complexity, is of considerable interest. Conventional modelling and prediction methods involve the construction of mathematical models describing the dynamic systems to be controlled and the application of analytical techniques to the model to derive prediction and control laws (Caudell, Xiao, & Healy, 2003; Kosko, 1992; Medsker, 1995; Rakovic, 1977; Vidyasagar, 1978; Wang, Devabhaktuni, & Zhang, 1998; Zadeh, 1965, 1973, 1994;). These models work well provided the system does meet the requirements and assumptions of synthesis techniques. However, due to uncertainty or sheer complexity of the actual dynamic system, it is very difficult to ensure that the mathematical model does not break down.

Neural network technology is an active research area (Chester, 1993; Grossberg, 1988; Kosko, 1992). It has been found useful when the process is either difficult to predict or difficult to model by conventional methods. Neural network modelling has numerous practical applications in control, prediction, and inference.

Time series are a special form of data where past values in the series may influence future values, depending on the presence of underlying deterministic forces. These are trend cycles and nonstationary behaviour in the time-series data are used in predictive models. Predictive models attempt to recognise patterns and nonlinear relationships in the time-series data. Due to the nature of data in time series, linear models are found to be inaccurate and there has been a great interest in nonlinear modelling techniques.

Recently, techniques from artificial-intelligence fields such as neural networks (NNs), fuzzy logic (FL), and genetic algorithms (GA) have been successfully used in the place of the complex mathematical systems for forecasting of time series (Azoff, 1994; Bauer, 1994; Cox, 1993, 1994; Davis, 1991; Gallant, 1993; Goldberg, 1989; Karr, 1991; Lee, 1990; Lee & Takagi, 1993; Mamdani, 1993; Michalewicz, 1992; Ruelle, 1989; Schaffer, 1994). These new techniques are capable of responding quickly and efficiently to the uncertainty and ambiguity of the system.

Neural networks (Azzof, 1994; Chester, 1993; Gallant, 1993; Hung, 1993; Karr, 1994; Knigham, 1996; Kingham, & Mohammadian, 1996; Welstead, 1994; Zuruda, 1994;) can be trained in an adaptive manner to map past and future values of a time series and thereby extract hidden structure and relationships governing the data (Lapedes, & Farber, 1987).

Investors and governments alike are interested in the ability to predict future interest-rate fluctuations from current economic data. Investors are trying to maximise their gains on the capital markets, while government departments need to know the current position of the economy and where it is likely to be in the near future for the well being of a country's people (Madden, 1995).

In the next section, the development of a hierarchical neural network system is considered. This section also describes the financial data that can be used to predict the fluctuations of interest rates in Australia. The application of hierarchical neural network systems for the prediction of quarterly interest rates in Australia is then considered.

Comparison of the results from single neural networks and the proposed hierarchical-neural network-system is made. The long-term prediction of interest rates by increasing

the forecast time to first 6-month then 1-year time periods is also considered. A hierarchical neural network system for predicting 6-month then 1-year time periods is compared with a traditional neural network system. Results of simulations are presented and conclusions and further-research directions are given in the last section of the chapter.

Neural Networks for the Prediction of Interest Rates

To predict fluctuations in the interest rate, a neural network system was created. There are a number of steps to perform to create the neural network system:

1. Identify the inputs and outputs for neural network system.
2. Preprocess data if required, and split into training and test suites.
3. Create a neural network system to predict the interest using training data.
4. Use the developed neural network on test data to evaluate the accuracy of the prediction of the system.

Identify the Inputs and Outputs for Neural Network System

To design a neural network system, the actual inputs and outputs must first be determined. There are a number of possible indicators that could be used to predict the interest rate. Some of the main economic indicators released by the Australian Government are:

- **Interest Rate**, which is the indicator being predicted. The interest rate used here is the Australian Commonwealth government 10-year treasury bonds.
- **Job Vacancies** are where a position is available for immediate filling or for which recruitment action has been taken.
- **The Unemployment Rate** is the percentage of the labour force actively looking for work in the country.
- **Gross Domestic Product (GDP)** is an average aggregate measure of the value of economic production in a given period.
- **The Consumer Price Index (CPI)** is a general indicator of the rate of change in prices paid by consumers for goods and services.

- **Household Saving Ratio** is the ratio of household income saved to a household's disposable income.
- **Home Loans** measure the supply of finance for home loans, not the demand for housing.
- **Average Weekly Earnings** is the average amount of wages that a full-time worker takes home before any taxes.
- **Current Account** is the sum of the balances on merchandise trade, services trade, income, and unrequited transfers.
- **Trade Weighted Index** measures changes in Australian currency relative to the currencies of our main trading partners.
- **RBA Commodity Price Index** provides an early indication of trends in Australia's export Prices.
- **All Industrial Index** provides an indication of price movements on the Australian Stock Market.
- **Company Profits** are defined as net operating profits or losses before income tax.
- **New Motor Vehicles** is the number of new vehicles registered in Australia.

The current interest rate is included in the input indicators to the system as the predicted interest rate is highly dependent on the current rate as there is only likely to be a small fluctuation in the interest rate. The current interest rate gives the neural network system an indication as to the expected "ball park" area of the predicted rate.

Preprocess Data

In most time-series predictions, there is some preprocessing of the data so that it is in a format that the system can use. This may be where data is normalised so it fits within certain boundaries, formatted into an appropriate form for the neural network system to use. It is also where decisions on how the data is represented are made.

There are a number of ways in which the raw data from the above indicators could be represented. Firstly, the system could just use the data "as is" and make its predictions from that. Alternatively, the system could instead use the difference from the previous quarter to the current quarter. The system could also take into consideration the effects of inflation on the raw data and compensate appropriately.

In our system, the change from one quarter to the next is used for the GDP and CPI indicators, while the interest rate is the actual reported rate from the Australian Bureau of Statistics.

Once the data has been preprocessed, it must be split into some groups for the training and testing of the system. For this system, the first two-thirds of the data was assigned to the training set while the other one-third was assigned to the test set. The system uses

the training set to train the neural network system. The neural network system is then tested on the test set.

Hierarchical Neural Network for Prediction of Interest Rates

In this section, a neural network is created to predict the following quarter's interest rate in Australia. The system uses the economic indicators described earlier. A hierarchical neural network system is used to predict the following quarter's interest rate. The first model uses the structure as shown in Figure 1, where the input parameters are split into a number of smaller related groups and their output is fed into the final group, which then produces the final interest-rate prediction. The second model used a single neural network system where all the input parameters were presented to the system and an interest-rate prediction was made.

In order for the neural network to use the economic data for predicting the following quarter's interest rate, a number of preprocessing steps must be performed. This allows the data to be presented to the neural network in a format with which it can easily work. Data presented to the neural network must fall within certain ranges (usually 0 to +1 or -1 to +1 (Rao & Rao, 1994)) due to the fact that the network uses a Sigmoid Activation function in its middle (or hidden) layers.

The neural network system formats the data for processing where the difference from the current quarter to the previous quarter is used as the data for the input. The change from one quarter to the next is used by all the indicators except the interest rate, where the actual interest rate is used. For example the Gross Domestic Product (GDP) would be formatted as shown in Table 1.

As Table 1 shows, there can still be a large range between the smallest and largest values. To reduce this into a more useable range, the data is modified by the following equation:

$$\text{New Data} = (\text{current data} - \text{Mean}) / \text{standard deviation} \quad (1)$$

The new data that has been calculated represents the distance from the mean value as a fraction of the standard deviation. This gives a good variability to the data, with only a few values that are out of the 0 to +1 or -1 to +1 range.

Table 1. Difference in data from current quarter to previous quarter

Year	Quarter	Data	Difference
1986	1	79856.0	N/A
1986	2	79520.0	-336.0
1986	3	79619.0	99.0
1986	4	79319.0	-300.0
1987	1	80201.0	882.0

The next step in the preprocessing stage is to squash the data so that it falls between the required range of 0 to +1 for this simulation. For this system, a Sigmoid squashing function is performed. The equation for this step is:

$$\text{Squash data} = 1 / (1 + \exp(-\text{Norm Data})) \quad (2)$$

After performing the sigmoid squashing function on the data, all the values fall in the range 0 to +1.

As well as using the indicators as inputs for the neural network, we also present data to the system that relates to the rate of change in the data, which is the second derivative of the data set (Rao & Rao, 1994). This accents changes in the data set between one quarter and the next. The equation for this is:

$$\text{mov diff} = (\text{current val} - \text{previous val}) / (\text{current val} + \text{previous val}) \quad (3)$$

The above equation is performed on the original data (before any preprocessing steps are performed) and will give a value between -1 and 1. The result from this equation becomes an additional input to the system. Therefore, for each input into the system, an additional input is created, doubling the number of input parameters to the neural network.

Hierarchical Neural Network System

In this section, we develop a number of neural network systems. These are then combined into a hierarchical neural network system to predict interest rates. Specifically, we look at why a hierarchical neural network system is important compared to a single neural network system and compare the results from single neural network systems with a hierarchical neural network.

The hierarchical neural network structure is formed by having the most influential inputs as the system variables in the first level of the hierarchy, the next important inputs in the second layer, and so on.

The first level of the hierarchy gives an approximate output, which is then modified by the second level and so on. This is repeated for all succeeding levels of the hierarchy. One problem occurs when it is not known which inputs to the system have more influence than the others. This is the case when using the economic indicators discussed earlier. Statistical analysis could be performed on the inputs to determine which ones have more bearing on the interest rate, however, without the advise of a statistician, it may be difficult to decide which statistical method to use.

The method used in this chapter is to split the inputs into a number of related groups. These inputs in these groups are related because they have some common connection

between the inputs, such as dealing with employment, or imports and exports. This changes the hierarchy into a two-level hierarchy, with the outputs from all the groups in the top layer giving their results as inputs into the bottom layer (Figure 1). Using the economic indicators already indicated we can develop five separate groups. These groups are as follows:

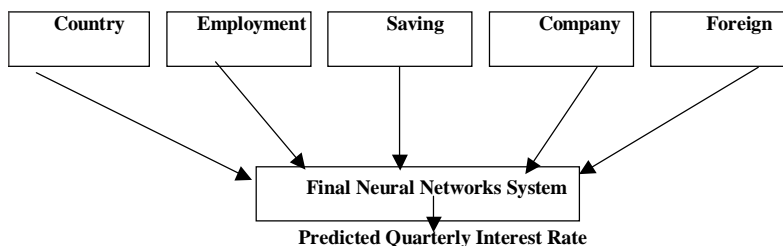
1. **Country Group** — This group contains Gross Domestic Product and Consumer Price Index.
2. **Employment Group** — This group contains the Unemployment Rate and the Job Vacancies indicators.
3. **Savings Group** — This group contains Household Saving Ratio, Home Loans, and Average Weekly Earnings.
4. **Company Group** — This group contains All Industrial Index, Company Profit, and New Motor Vehicles indicators.
5. **Foreign Group** — This group contains Current Account, Trade Weight Index, and also the RBA Commodity Index.

These five groups each produce a predicted interest rate for the next quarter. These are then fed into the next layer of the hierarchy where the final predicted interest rate is found, as shown in Figure 1. For each of these groups, the current quarter's interest rate is included in the indicators used. The current interest rate has the biggest influence on the following quarters' interest rates.

The five neural network systems created form the top layer of the hierarchy. They are connected together to form a final neural network system. The final neural network system uses the predicted interest rate from the five above groups to produce a final interest-rate prediction.

In the following sections, we first create each of the neural network systems required for the top layer of the hierarchy. We then combine first four and then all five groups together to form the final hierarchical neural network system to predict the quarterly interest rate in Australia.

Figure 1. Hierarchical neural network system for interest-rate prediction



Country Neural Network

The country neural network contains information relating to countries current economic performance. These indicators are: Consumer Price Index and Gross Domestic Product. As a measure of how well a group predicts the following quarters' interest rates, we calculate the average error of the system for the training set and tests sets. This is calculated using the following formula:

$$E = \frac{\sum_{i=1}^n abs(P_i - A_i)}{n} \quad (4)$$

where E is the average error, P_i is the predicted interest rate at time period i , A_i is the actual interest rate for the quarter, and n is the number of quarters predicted.

The Table 2 shows the results for the average error for country group.

Table 2 shows the training-average error is less than the test-average error as the test set uses data that has not been used in the training of the neural network system.

Company Neural Network

The company neural network contains information relating to the corporate sector of the market. This information includes: All Industrial Index, Company Profit, and New Motor Vehicle Registrations. These three indicators, combined with the Interest Rate, are used to predict the following quarter's interest rate.

Using the training data, the system is able to predict the following quarters' interest rates with only a few fluctuations from the actual interest rate. However, the same problems occur when predicting the interest rate on the test data.

The average error of the company group of indicators is shown in Table 3. It shows that there was a slight decrease in the average error of the simulation when compared to the

Table 2. Average error for neural network country group

Training Average	Test Average	Overall Average
0.401	1.023	0.591

Table 3. Average error for neural network company group

Training Average	Test Average	Overall Average
0.228	1.290	0.548

country group. However, the test-set average error is larger than that of the country group.

Employment Neural Network

The employment neural network contains information relating to the employment sector of the economy. This information includes: Unemployment Rate and Job Vacancies.

These two indicators, combined with the Interest Rate, are used to predict the following quarters' interest rates. Using the training data, the neural network system is able to predict the following quarter's interest rate, with some fluctuations from the actual interest rate. The performance of the neural network system for predicting the interest rate on the test data looks slightly better than that achieved by the neural network in the country or company groups. The average error of the employment group of indicators is shown in Table 4.

Savings Neural Network

The savings neural network contains the following indicators: Savings Ratio, Home Loan approvals, and Average Weekly Earnings. These three indicators, combined with the Interest Rate, are used to predict the following quarter's interest rate. The neural network system for the savings group has a number of error amount peaks up as high as 2%, however during the training period, there are only the two peaks, with the others happening during the test period. This compares well with the other groups looked at so far, as while there are some fairly large error amounts, there are a number of quarters where there is a very low amount of error.

The average error of the savings group of indicators is shown in Table 5.

Table 4. Average error for neural network employment group

Training Average	Test Average	Overall Average
0.352	0.742	0.471

Table 5. Average error for neural network savings group

Training Average	Test Average	Overall Average
0.378	0.923	0.534

Table 6. Average error for neural network foreign group

Training Average	Test Average	Overall Average
0.387	0.714	0.487

Foreign Neural Network

The foreign neural network contains information relating to Australia's current economic position in relation to the rest of the world. The indicators used are: Current Account, Reserve Bank of Australia Commodity Price Index, and Trade Weight Index. These three indicators, when combined with the Interest Rate, are used to predict the following quarter's interest rate. The neural network system for this group has a number of fluctuations in interest rate that are not accurately predicted by the neural network system. In one quarter (quarter 46), there is a difference between the actual and predicted interest rate of more than 3%. However, the rest of the quarters perform better than this and compare favourably with previously generated neural network system for other groups.

The average error of the foreign group of indicators is shown in Table 6. It shows that the training-average error amount is larger than that achieved by both the savings and Company groups.

Building a Hierarchical Neural Network System by Combining the Neural Network Systems for Each Group

After creating the above neural network system, we must combine them so that we can utilise the information they present and obtain better predictions of each quarter's interest rate than any of the neural network systems previously created.

Combining first four and then all five of the neural network systems from the previous section created a hierarchical neural network system. The way these groups were combined to form the hierarchical neural network system is illustrated in Table 7.

Table 7. Hierarchical neural network system groups

Combine four groups	Combine five groups (as in Figure 1)
company group	company group
country group	country group
employment group	employment group
savings group	savings group
	foreign group

Table 8. Comparison of neural network systems

	Average Training Error	Average Test Error	Average Overall Error
Four-group hierarchical neural network	0.318	0.681	0.428
Five-group hierarchical neural network	0.354	0.607	0.431
All indicators single neural network	0.039	0.880	0.296

A backpropagation neural network is used with two hidden layers, each consisting of 20 neurons; output layer consists of one node. This was found to produce a quicker and more accurate result than using a single hidden layer. Sigmoid learning is used to predict the following quarter's interest rate. The error tolerance was set to 0.0001, the Learning Parameter (Beta) was set to 0.6, momentum (alpha) and Noise Factor were both set to 0. The neural network was trained for 10000 cycles.

As Table 8 shows, the range of results for the different systems is very diverse. The training-average error, which is the average error, recorded during the training period of 40 quarters, ranges from a high value of 0.318 for the four groups down to 0.039 for the all-indicator neural network. The all-indicator neural network was able to learn the training data almost perfectly.

The test-average error is the average error recorded during the test period, which is where the system is presented with inputs that it has not been trained on. These compare favourably with all indicator neural network systems had disappointing test-average error results.

Long-Term Predictions

So far we have looked at predicting the following quarter's interest rate, which is 3 months from the current quarter. There are a number of situations in which this time period is too short a prediction length, such as when investors have to decide whether to move from the bond market into the property market before the end of the financial year.

In the next section, a hierarchical neural network system for predicting interest rates, 6 months (biyearly) from the current quarter is developed, followed by a hierarchical neural network system that predicts the interest rate one year ahead of the current quarter. The structure and grouping of the inputs and neural network systems in the following section for the hierarchical neural network system for long-term prediction is the same as the hierarchical neural network system described above for quarterly interest-rate prediction.

Table 10. Comparison of results for long-term predictions

6-Month Predictions	Training Error	Test Error	Overall Error
Four-group hierarchical neural network	0.196	0.794	0.378
Five-group hierarchical neural network	0.197	0.813	0.383
All indicators single neural network	0.079	1.421	0.480
1-Year Predictions			
Four-group hierarchical neural network	0.110	1.248	0.456
Five-group hierarchical neural network	0.139	1.114	0.435
All indicators single neural network	0.054	1.320	0.439

Comparison Between Neural Networks for Long-Term Predictions

From the figures given in the table below, it can be seen that the results for long-term predictions produce similar results to the short-term predictions (i.e. quarterly). Table 9 shows a comparison between the hierarchical neural network predictions and the single neural network predictions for 6 months and 1 year.

From these results, it can be seen that the hierarchical neural network systems have a much better test-average error when compared to the all indicator single neural network systems. The training-average error results were similar for most of the prediction systems, with only the all-indicator single neural network system, for both 6-month and 1-year predictions, which had very low training results. However, the average-test-error amounts for the all-indicator single neural network systems were the highest of all the systems.

From these results, we can conclude that using 14 economic indicators and training the system for 40 quarters, the hierarchical systems provide much better prediction results than the all-indicators single neural network systems. From the results obtained in Table 9, it can be seen that the average-training error results were similar for most of the prediction systems, with only the all-indicators single neural network system, for both 6-month and 1-year predictions, which had very low training results. However, the average test-error amounts for the all-indicator systems were the highest of all the systems.

From these results, we can conclude that using 14 economic indicators and training the system for 40 quarters, the hierarchical neural network systems provide much better prediction results than the all-indicators single neural network systems. These results are similar to the comparisons found in the previous section where the Interest Rate predictions for one quarter by the hierarchical neural networks system provided better prediction results than the all-indicator neural network system.

Conclusion and Further Research Direction

This chapter has presented a method in which a hierarchical neural networks system can be used to model and predict the fluctuations of the Australian Interest Rate using Australian economic data.

The application of the proposed method to modelling and prediction of interest rate using Australian economic indicators is considered.

From simulation results, it was found that the hierarchical neural networks system is capable of making accurate predictions of the following quarter's interest rate. The results from the hierarchical neural networks were compared to a neural network that used all the indicators as inputs.

Long-term predictions for 6 months and 1 year from the current quarter were then undertaken with the hierarchical neural network systems proving to be more accurate in their predictions than the neural network systems. These results were found to be similar to those obtained when quarterly interest rates were predicted.

Having a time lag for some economic indicators may increase prediction accuracy. There are some indicators whose effect is not felt on the interest rate for a number of quarters, such as Consumer Price Index (Larrain, 1991). Delaying the indicator results in the system using the indicator when it has more effect on the interest rate. The accuracy of the hierarchical neural network system may also be increased if an indicator that fluctuates greatly between quarters is smoothed out using some form of moving average (such as 2-quarter, or 6-month, moving average). This would then remove any sudden peaks (or valleys) that the indicator may exhibit which could greatly affect the prediction accuracy.

Finally the structure of the hierarchical neural network system may affect the performance of the system. This is the subject of our further research.

References

- Azzof, E. M. (1994). *Neural networks time series forecasting of financial markets*. New York: John Wiley & Sons.
- Bauer, R. J. (1994). *Genetic algorithms and investment strategies*. New York: John Wiley & Sons.
- Caudell, T. P., Xiao, Y., & Healy, M. J. (2003), ELOOM and Flatland: Specification, simulation and visualization engines for the study of arbitrary hierarchical neural architectures. In D. C. Wunsch, II, M. Hasselmo, K. Venayagamoorthy, & D. Wang (Eds.), *Advances in neural network research*.
- Chester, M. (1993). *Neural networks: A tutorial*. Englewood Cliffs, NJ: Prentice-Hall.
- Cox, E. (1993, February). Adaptive fuzzy systems. *IEEE Spectrum*, 27-31.

- Cox, E. (1994). *The fuzzy systems handbook: A practitioner's guide to building, using and maintaining fuzzy systems*. New York: Academic Press.
- Davis, L. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.
- Gallant, S. I. (1993). *Neural network learning and expert systems*. Cambridge, MA: MIT Press.
- Goldberg, D. (1989). *Genetic algorithms in search, optimisation and machine learning*. Reading, MA: Addison Wesley.
- Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks, 1*, 17-68.
- Hung, C. (1993). Building a neuro-fuzzy learning control system. *AI Expert*, 8(11), 40-49.
- Karr, C. (1991). Genetic algorithms for fuzzy controllers. *AI Expert*, 6(2), 26-33.
- Karr, C. (1994). In R. R. Yager, & L. A. Zadeh (Eds.), *Adaptive control with fuzzy logic and genetic algorithms, fuzzy sets and neural networks, and soft computing*. New York: Van Nostrand Reinhold.
- Kingham, M., & Mohammadian, M. (1996, November). Financial modelling and prediction of interest rate using fuzzy logic and genetic algorithms. In *4th Australian and New Zealand Intelligent Information Systems Conference (ANZIIS '96)*.
- Kosko, B. (1992). *Neural networks and fuzzy systems, a dynamic system*. Englewood Cliffs, NJ: Prentice-Hall.
- Kosko, B., & Isaka, S. (1993, July). Fuzzy logic. *Scientific American*, 76-81.
- Lapedes, A., & Farber, R. (1987). *Nonlinear signal processing using neural networks, prediction and system modelling* (Los Alamos Report No. LA-Ur-87-2662). Los Alamos, NM: Los Alamos National Laboratory.
- Larrain, M. (1991, September-October). Testing chaos and non linearities in T-bill rates. *Financial Analysts Journal*, 51-62.
- Lee, C. C. (1990). Fuzzy logic in control systems: Fuzzy controllers — Part I, Part II. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2), 404-435.
- Lee, M. A., & Takagi, H. (1993, July 17-22). Dynamic control of genetic algorithm using fuzzy logic techniques. In *Proceedings of the 5th International Conference on Genetic Algorithms* (pp. 76-83).
- Madden, R. (1995). *Measuring Australian economy* (Catalogue No. 1360.0). Australian Bureau of Statistics.
- Mamdani, E. H. (1993). Twenty years of fuzzy control: Experiences gained and lessons learnt. In *Proceedings of 2nd IEEE International Conference on Fuzzy Systems*, San Diego, CA (pp. 339-344).
- McClelland, J. L., & Rumelhart, D. E. (1988). *Explorations in parallel distributed processing*. Cambridge, MA: MIT Press.
- Medsker, L. (1995). *Hybrid intelligent systems*. Norwell, MA: Kluwer Academic.
- Michalewicz, Z. (1992). *Genetic algorithms + data structure = evolution programs*. Springer Verlag.

- Mohammadian, M., & Stonier, R. J. (1994, July 18-20). Tuning and optimisation of membership functions of fuzzy logic controllers by genetic algorithms. In *3rd International Workshop on Robot Human Communication RO-MAN'94*, Japan, Nagoya.
- Rakovic, D. (1997). In L. Rakic, G. Kostopoulos, D. Rakovic, & D. Koruga (Eds.), *Proceedings of ECPD Workshop (ECPD)* (pp. 189-204).
- Rao, V. B., & Rao, H. V. (1994). *C++ neural networks and fuzzy logic*. MIS Press.
- Refenes, A. (1995). *Neural networks in the capital markets*. New York: Wiley.
- Ruelle, D. (1989). *Chaotic evolution and strange attractors: The statistical analysis of time series for deterministic nonlinear systems*. Cambridge University Press.
- Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: Exploration in the microstructure of cognition*. MIT Press.
- Schaffer, J. D. (1994). Combinations of genetic algorithms with neural networks of fuzzy systems. In Zurada, Marks, & Robinson (Eds.), (pp. 371-382).
- Schwenker, F., & Kestler, H. A. (2001). 3-D visual object classification with hierarchical RBF networks. In R. J. Howlett, & L. C. Jain (Eds.), *Radial basis function neural networks 2: New advances in design* (pp. 269-294). Physica-Verlag.
- Takagi, T., & Sugeno, M. (1983). Derivation of fuzzy control rules from human operator's control actions. In *Proceedings of the IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis* (pp. 55-60).
- Vidyasagar, M. (1978). *Nonlinear systems and analysis*. Englewood Cliffs, NJ: Prentice Hall.
- Wang, F., Devabhaktuni, V., & Zhang, V. (1998). Hierarchical neural network approach to the development of library of neural models for microwave design. *IEEE Transaction Microwave Theory and Techniques*, 46, 2391-2403.
- Welstead, T. (1994). *Neural networks and fuzzy logic applications in C/C++*. New York: Wiley.
- Zadeh, L. (1965). Fuzzy sets. *Inf. Control*, 8, 338-353.
- Zadeh, L. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transaction Syst. Man. Cybern.*, SMC-3, 28-44.
- Zadeh, L. (1994). Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*, 37(3), 77-84.

Chapter VII

Forecasting the Term Structure of Interest Rates Using Neural Networks

Sumit Kumar Bose, Indian Institute of Management, India

Janardhanan Sethuraman, Indian Institute of Management, India

Sadhalaxmi Raipet, Indian Institute of Management, India

Abstract

The term structure of interest rates holds a place of prominence in the financial and economic world. Though there is a vast array of literature on the issue of modeling the yield curve, there is virtually no mention of the issue of forecasting the yield curve. In the current chapter, we apply neural networks for the purpose of forecasting the zero-coupon yield curve. First the yield curve is modeled from the past data using the famous Nelson-Siegel model. Then, forecasting of the various parameters of the Nelson-Siegel yield curve is done using two different techniques: the multilayer perceptron and the feed-forward network. The forecasted Nelson-Siegel parameters are then used to predict the yield and the price of the various bonds. Results show the superiority of the feed-forward network over the multilayer perceptron for the purposes of forecasting the term structure of interest rates.

Introduction

The term structure of interest rates is a relation of the yield and the maturity of default-free zero-coupon securities and provides a measure of the returns that an investor might expect for different investment periods in a fixed income market. The term structure of interest rates is a topic of central importance in economic and financial theory. As a result, the modeling and estimation of the term structure has received considerable attention of a number of researchers right from the early sixties. Broadly speaking, there are two popular approaches for modeling the term structure of interest rates: a) fitting curves to the data using standard statistical techniques and, b) dynamic asset-pricing method. The parsimonious representation dictated by an exponential decay term such as Nelson and Siegel (1987), Svensson (1994) and the spline representation categorized into parametric and nonparametric splines such as Adams and van Deventer (1994), Fama and Bliss (1987), Fisher, Nychka, and Zervos (1995), McCulloch (1971, 1975), McCulloch and Kwon (1993), Tanggaard (1997), Vasicek and Fong (1982), and Waggoner (1997) belong to the former approach of estimating the term structure. While Vasicek and Fong (1982) explore the possibility of using exponential splines, McCulloch (1975) explores the possibility of fitting parametric cubic splines. Dynamic asset pricing method of estimating the term structure includes no-arbitrage models of Heath, Jarrow, and Morton (1992), Ho and Lee (1986), Hull and White (1990) and the various equilibrium models such as the affine general equilibrium models, for example the model of Pearson and Sun (1994). Affine models hypothesize yield as affine function of the state variables and include the models of Cox, Ingersoll, and Ross (1985) and Duffie and Kan (1996) apart from others. In spite of a flurry of research activity on modeling the yield curve, there has been little research effort on forecasting the yield curve (Diebold & Li, 2002). Forecasting the term structure of interest rates is important from the viewpoint of investment decisions of firms, saving decisions of consumers, policy decisions of governments, pricing and hedging decisions of derivatives, valuation decisions of various financial products especially the debt instruments and managing the bond portfolio apart from a host of other decisions. No-arbitrage models are applicable only at a particular time slice as their focus is on fitting the cross section of interest rates at a particular time. The models therefore fail to capture the time-series dynamics. These models are hence not very useful for forecasting purposes. The equilibrium models, on the other hand, are able to capture the time-series dynamics, but fail to pay attention to fitting the cross section of interest rates at any given time. Though the equilibrium models are better candidates in contrast to the no-arbitrage models for forecasting purposes, the forecasts generated by the equilibrium models have been shown to be extremely poor. Most models so far in the literature — including the no-arbitrage models and the equilibrium models — fail to model the dynamic relationship between the parameters of a term-structure model. Models of McCulloch (1993), Nelson-Siegel, and others try to explain the movements of the term structure with the aid of various factors and consequently attach labels to these factors having important macroeconomic and monetary policy underpinnings. Others such as Pearson and Sun (1994) interpret the factors in their model as “short rate” and “inflation,” and Litterman and Scheinkman (1991) interpret the factors used in the model as “level,” “slope,” and “curvature.” The labels attached to the factors stand for the influence the factors have

on the yield curve and describe how the yield curve shifts or changes shape in response to the changes in the factor. The forecasting ability of most of the existing term-structure models, however, is poor. This has motivated us to try out the neural network theory for the purposes of forecasting the yield curve in the present research.

Following the work of Diebold and Li (2002), in the present paper we adopt the Nelson-Siegel (1987) framework to model the yield curve in each period. We then design a neural network model to directly model the relationships between the time varying parameters in each period.

The second section discusses the appropriateness of neural networks for financial modeling. The third section describes the dynamics of the yield curve. The Nelson-Siegel method of modeling the term structure and the interpretation of the various Nelson-Siegel parameters are also provided. The fourth section describes the methodology and the neural network architecture in detail. The fifth section discusses the results. Conclusion and various issues associated with the application of the neural networks are explained in the sixth section.

Why Neural-Network-Based Models are Appropriate

Traditional forecasting techniques such as regression analysis, moving averages, and smoothing methods require assumptions about the form of the population distribution. For example, the regression models assume that the underlying population is normally distributed. On the other hand, the neural network models (Hassoun, 1995) do not require any such assumption about the distribution for the underlying population. Moreover, the various factors in the economy interact in a number of complex ways to affect the yield curve. The complexity of interactions is hard to model mathematically and the relationships between the variables underlying these factors are more likely to be nonlinear than linear. Thus assuming a mathematical model a-priori, may provide an oversimplistic picture of the interactions than are actually present. Such nonlinear relationships amongst the various factors make it difficult for the traditional methods to discern any meaningful relationship between the factors, whereas neural networks are able to infer such complex nonlinear relationships between the effected variable and its determinants. Activities undertaken in the financial and economic systems, though designed by humans, are highly complex. The complexity arises from the different expectations and diverse reactions of the different players in the market. Such complex systems are difficult to capture in terms of mathematical equations and hence neural-network-based models seem appropriate. One of the most successful applications of neural networks in the finance literature is of Hutchinson, Lo and Poggio (1994). They have shown that the Black-Scholes formula can be successfully learnt using the neural network models. Furthermore, they have demonstrated the ability of the neural network models to successfully learn a relationship between the option and the underlying risk factors.

Yield-Curve Dynamics and the Nelson-Siegel Method

Here we introduce the fundamentals of the yield curve. We also build the framework for modeling and forecasting the yield curve, where we argue in favor of the appropriateness of the Nelson-Siegel method for the purposes of modeling the yield curve.

If $PV_t(\tau)$ represents the price of a τ period discount bond and $y_t(\tau)$ represents its continuously compounded zero-coupon nominal yield to maturity (YTM), then the discount curve can be obtained from the yield curve as: $P_t(\tau) = e^{-\tau y_t(\tau)}$. We can then derive the instantaneous forward rate curve from the discount curve as $f_t(\tau) = -P_t'(\tau) / P_t(\tau)$. The relationship between the yield to maturity and the forward rate is therefore given as:

$$y_t(\tau) = \int_0^{\tau} \frac{f_t(u) du}{\tau}$$

or,

$$f_t(\tau) = y_t(\tau) + \tau y_t'(\tau)$$

This means that the zero-coupon yield is an equally weighted mean of the forward rates. It is therefore possible to price any coupon bond as the sum of the present values of future cash flows (coupon payments and the principal payment), if the yield curve or the forward-rate curve is given.

Nelson-Siegel Yield Curve and Its Interpretation

Nelson-Siegel models the instantaneous forward-rate curve as:

$$f_t(\tau) = \beta_{0t} + \beta_{1t} e^{-\lambda_t \tau} + \beta_{2t} \lambda_t \tau e^{-\lambda_t \tau}.$$

This implies that the equation of the Nelson-Siegel yield curve is given as:

$$y_t(\tau) = \beta_{0t} + \beta_{1t} \left(\frac{1 - e^{-\lambda_t \tau}}{\lambda_t \tau} \right) + \beta_{2t} \left(\frac{1 - e^{-\lambda_t \tau}}{\lambda_t \tau} - e^{-\lambda_t \tau} \right).$$

Nelson-Siegel forward-rate curve can be interpreted as laguerre function plus a constant. Laguerre function is a highly popular mathematical approximating function and is polynomial times an exponential decay term. The rate of exponential decay is governed by the parameter λ_t . Large values of λ_t produce faster decay and can fit the curve at short maturities; small values of λ_t produce slower decay and better fit the curve at longer maturities. β_{0t} , β_{1t} , β_{2t} can be interpreted as three latent factors. β_{0t} can be interpreted as the long-term factor, as the loading on β_{0t} is 1, which is a constant and does not change

with time. β_{1t} can be interpreted as the short-term factor as the loading on β_{1t} is $\left(\frac{1-e^{-\lambda_t\tau}}{\lambda_t\tau}\right)$,

a function that starts at 1 and in the limit decreases to zero exponentially and monotonically. β_{2t} can similarly be interpreted as the medium-term factor as the loading on β_{2t} that

is, $\left(\frac{1-e^{-\lambda_t\tau}}{\lambda_t\tau} - e^{-\lambda_t\tau}\right)$ first increases from 0 and then drops down to 0. Thus, the three factors

β_{0t} , β_{1t} , β_{2t} govern three important aspects of the yield-curve level, slope, and curvature and can therefore appropriately be called level, slope, and curvature respectively instead of long-term, short-term, and medium-term.

Justification for Using the Nelson-Siegel Model

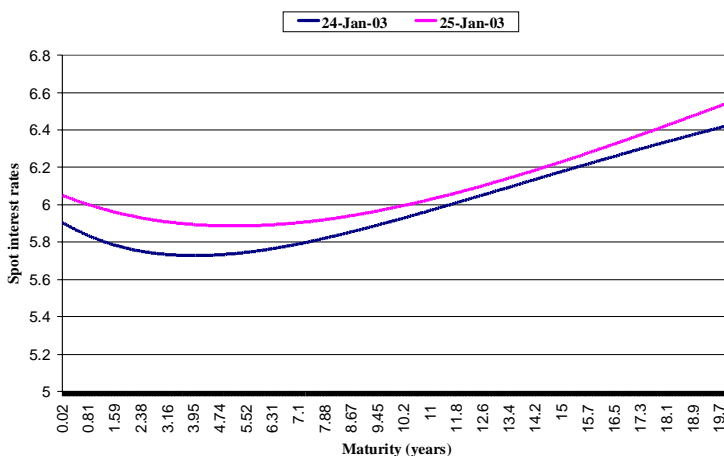
The justification of using the Nelson-Siegel exponential functional form over the spline-based methods is many. First, because of its parsimony, its users are able to remove noise from the data and avoid overfitting of risk. This allows identification of only the salient features of the bond dataset and at the same time avoids fitting the random features of the data set that may not recur. Second, the empirical analysis by Litterman and Scheinkman (1991) has shown that three factors are sufficient to completely explain the dynamics of the term structure of interest rates. Third, the number of parameters to be estimated in the Nelson-Siegel framework is much less than that in a spline-based approach. Fourth, the assumption of a unique functional form for the discount function over the complete range of maturities allows some of the fundamental properties of the discount function to be imposed a-priori. Fifth, comparative assessment of many a past research works has shown that Nelson and Siegel and its extension by Svensson (1994), are better performers than their spline counterparts. Sixth, the assumption of a unique functional form for the discount function automatically imposes the no-arbitrage condition because it is possible to obtain the discount function only if the assumption of no arbitrage holds. The functional form postulated by Nelson and Siegel is able to accommodate diverse shapes of the forward rate curves like the monotonic and humped shapes. Moreover it provides an intuitive explanation of the parameters: β_0 is the long rate, β_1 is the short rate and β_2 is the weight attached to the medium rate. This allows us to forecast long- and short-rate movements. Also, as suggested by Nelson and Siegel, the existing framework provides a solid ground for generalization to higher-order models.

Methodology

In this chapter, we adopt the Nelson-Siegel model for fitting the yield data. Once the Nelson-Siegel exponential form has been selected as the approximating function, the parameters of the function have to be estimated using one of the several approaches such as maximum likelihood, iterative programming, weighted least squares, linear programming amongst a host of other approaches. In this chapter, we adopt the method of least squares. We estimate the parameters β_{0t} , β_{1t} , β_{2t} in the manner illustrated in the original work of Nelson and Siegel. For each day t we fix the value of λ_t and compute the values of the two regressors, that is the factor loadings, and then apply the least-square method to estimate β_{0t} , β_{1t} , β_{2t} . This is repeated for different values of λ_t , and that value of λ_t is retained for which the error is least. When we apply the least square method to the yield data of each day, we get the time series of the estimates of β_{0t} , β_{1t} , β_{2t} . Figure 1 shows the modeled Nelson-Siegel yield curve for the two selected dates. It is seen from the diagram that the three-factor Nelson-Siegel yield curve can sufficiently replicate the different shapes of the yield curve.

The values of the parameters obtained by fitting the Nelson-Siegel curve to the yield data of the previous day is fed to a neural network for forecasting the parameters of the Nelson-Siegel curve for the coming day. This is then used to predict the yield and hence the price of the bond.

Figure 1. Plot of the term structure (zero-coupon yield curve) for dates 24 January 2003 and 25 January 2003



Modeling Yield Curves

As already mentioned, we fit the yield data using Nelson-Siegel's three-factor model.

$$y_t(\tau) = \beta_{0t} + \beta_{1t} \left(\frac{1 - e^{-\lambda_t \tau}}{\lambda_t \tau} \right) + \beta_{2t} \left(\frac{1 - e^{-\lambda_t \tau}}{\lambda_t \tau} - e^{-\lambda_t \tau} \right).$$

The reason for choosing the Nelson-Siegel model for fitting the yield curve as cited earlier is the natural interpretation of the three beta parameters of the curve and the easiness with which it can model the different shapes of the curve. It is parsimonious and has a discount function that starts at zero and in the limit approaches zero. Bliss (1997) made a comparison of the different yield-curve fitting methods and noted that the Nelson-Siegel method outperforms most other methods. Moreover Diebold and Li (2002) show the ability of the Nelson-Siegel model to replicate the various stylized facts of the yield curve and the inability of affine models to do the same (Duffee, 2002). Hence, the Nelson-Siegel method is a natural choice for modeling the yield curve.

Forecasting the Yield-Curve Parameters

Neural Network Architecture

The Nelson-Siegel parameters viz β_0 , β_1 , β_2 and λ in period $t-1$ are defined as inputs and the Nelson-Siegel parameters in period t are defined as the output to which the neural network maps the inputs. After sufficient training (MSE of 0.01 or 1000 epochs whichever is earlier), the network can be used to predict yield for an out-of-sample period. We consider models based on two different network architectures — the first based on multilayer perceptron and the second based on feed-forward networks.

Multilayer Perceptron

The perceptron is the simplest neural network and consists of a single input layer and a single output layer. The perceptron is a feed-forward network where neurons in any layer are only connected to successor layers. This means there cannot be any connections from one layer to layers other than the adjacent ones. The multilayer perceptron can contain an arbitrary number of hidden layers. Input patterns propagate through the multilayer perceptron using the same feed-forward algorithm as the perceptron. The main difference is the addition of a weight matrix between each hidden layer or between a hidden layer and the output layer. As opposed to a threshold activation function, many multilayer perceptrons use an alternate activation function known as the "Linear Tanh" activation function. The Linear Tanh Axon substitutes the intermediate portion of the tanh by a line of slope b , making it a piecewise linear approximation of the tanh.

Backpropagation learning is a popular learning algorithm that is used in many multilayer perceptrons. The algorithm creates an error function that is defined over the entire possible space of the weights for the network. This global minimum represents an error value of zero and would correspond to the ideal weight values for a given input pattern. Weights are updated by following the steepest slope or gradient of the error function. However, a problem arises when local minima are present. The algorithm may find a local minimum value as opposed to the global minimum leaving the optimal weight values unobtainable. Thus, the weights that may be obtained after the termination of the algorithm may be substantially different from the desired global minima that capture the desired features of the underlying problem. This problem is particularly acute when the error surface is highly uneven. We therefore use a variation of the backpropagation learning algorithm called the *momentum learning rule* (Haykin, 1999), with an additional *momentum* term in the weight updation rule. Multilayer perceptrons have been widely used in various financial-forecasting techniques like the currency exchange-rate predictions, gilt futures pricing, and so on (Refenes, Azema-Barac, Chen, & Karoussos, 1993).

Generalized Feed-Forward Network

Generalized feed-forward networks are a generalization of the MLP such that connections can jump over one or more layers. In theory, an MLP can solve any problem that a generalized feed-forward network can solve. In practice, however, generalized feed-forward networks often solve the problem much more efficiently. A classic example of this is the two-spiral problem (Whitley & Karunanithi, 1991).

Learning Algorithm

We used the momentum learning rule (Haykin, 1998), a supervised learning rule, for both MLP as well as feed-forward training. The momentum learning is similar to the backpropagation algorithm (BP). The key idea is to present the input vector to the network; calculate in the forward direction of the output of each layer and the final output of the network. For the output layer, the desired values are known and therefore the weights can be adjusted as is done in the case of single layer network. To calculate the weight changes in the hidden layer, the error in the output layer is backpropagated to these layers according to their connecting weights. This process is repeated for each sample in the training set. Momentum learning is an improvement over the gradient descent search in the sense that a memory term (the past increment to the weight) is used to speed up and stabilize convergence (Rao & Principe, 2000). The weight increment is then adjusted to include some fraction a of the previous weight update; therefore it becomes:

$$w(m+1) = w(m) + (1-a)Dw(m) + aDw(m-1)$$

Where $Dw(m-1) = w(m) - w(m-1)$ is the past weight change.

The fraction a should not be negative, and for stability it must be less than 1.0. Normally, it should be set between 0.5 and 1.0. If $a = 0$, the algorithm is reduced to the standard backpropagation. This rule is called momentum learning due to the form of the last term, which resembles the momentum in mechanics. For the current experiment, we chose a value of 0.7 for the momentum parameter a . Thus, the weights are changed proportionally to how much they were updated in the last iteration. Therefore, if the search is going down the hill and finds a flat region, the weights are still changed, not because of the gradient, but because of the rate of change in the weights.

Data Samples: Testing and Training Sets

The data of different bonds maturing at various time intervals offering different rate of returns for the purpose of conducting the experiments is taken from NSE (National stock exchange, India).

- **Training-Data Set** — The training-data set consisted of 600 Nelson-Siegel parameters (300 in years 2001 & 2002 and 300 in years 2003 & 2004).
- **Testing-Data Set** — The testing-data set consisted of 60 betas (30 in years 2001 and 2002 and 30 in years 2003 and 2004). Additionally, 20 days in 2004 were selected at random from the test-data set, and the forecasted betas corresponding to these days were used to forecast the prices for various different bonds. An average 20 different bonds were taken on each of the 20 days (thus 400 bond instances) for the previously mentioned testing purposes.

Error Measures of Prediction

Two different measures were considered for assessing the quality of forecast. The first one is the mean-square error (MSE), and the second measure is the percentage error in prediction. Mean-squared error is calculated as the mean of squares of the difference between market price and the predicted price. Thus:

$$MSE = \left\{ \frac{\Sigma(\text{Market Price of the bond} - \text{Predicted Price})^2}{\text{Total number of test samples}} \right\}$$

Percentage error in prediction is calculated by the following formula:

$$\% \text{ error in Prediction} = \left\{ \frac{(\text{Market Price of the bond} - \text{Predicted Price})}{\text{Market Price of the bond}} \right\} * 100$$

Results

Several experiments were conducted with various architectures of MLP & feed-forward networks. After a number of experiments, the number of hidden layers in both the cases was fixed at one. Each hidden layer consisted of four processing units. Theoretically, it has been shown that MLPs with a wide variety of continuous hidden-layer activation functions, one hidden layer with an arbitrarily large number of units suffices for the “universal approximation” property (Hornik, 1993; Hornik, Stinchcombe, & White, 1989).

Fit of the parameters, that is $\beta_0, \beta_1, \beta_2, \lambda$ during the testing phase of MLP are depicted in Figure 2(a)-(d). Similar diagrams can be shown for the feed-forward network, though they are purposefully avoided. Table 1 shows the average error in prediction of $\beta_0, \beta_1, \beta_2$ and λ with actual values modeled by Nelson-Siegel method on the test-data set. However, what matters most is the error generated in forecasting the bond price calculated using

Figure 2(a). Variation between actual and neural network values of β_0 on out-of-sample data (MLP)

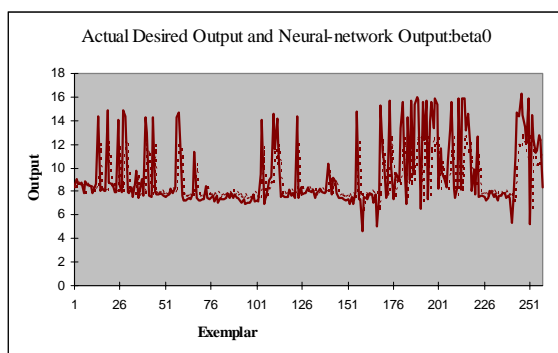


Figure 2(b). Variation between actual and neural network values of β_1 on out-of-sample data (MLP)

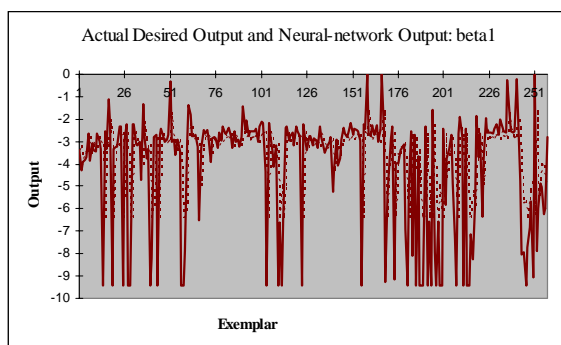


Figure 2(c). Variation between actual and neural network values of β_2 on out-of-sample data (MLP)

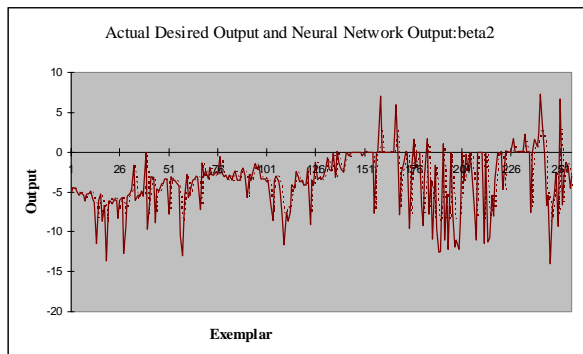


Figure 2(d). Variation between actual and neural network values of λ on out-of-sample data (MLP)

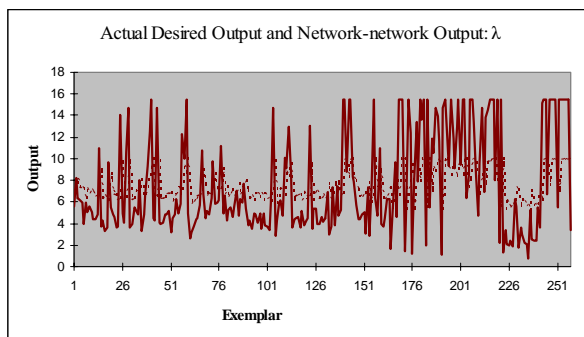


Table 1. Average percentage error in prediction of β_0 , β_1 , β_2 and λ using MLP and feed-forward architectures

Parameters	Average Percentage Error	
	(MLP)	(Feed forward)
β_0	7.09128	6.851593
β_1	6.00752	5.86612
β_2	13.59411	13.05343
λ	16.85239	16.91081

Table 2. Mean-square error in prediction of bond price

Multilayer Perceptron	Feed-forward Network
7.377	4.094

Table 3. Average percentage error in prediction of bond price

Multilayer Perceptron	Feed-forward Network
0.00709	-0.00023

the forecasted values of the Nelson-Siegel parameters of the yield curve. So, in some sense, the comparison of the forecasted Nelson-Siegel parameters with the modeled Nelson-Siegel parameters on the test data is of only of pseudoimportance.

Tables 2 and 3 give the MSE and the average percentage error in prediction of bond prices for both MLP and feed-forward networks. Comparative performance of the feed-forward networks is better than the MLP. The model based on a feed-forward network is better able to capture the diverse facets of the term structure than the model based on multilayer perceptron.

The models where we make use of the Nelson-Siegel method along with the neural network models produce significantly fewer pricing errors and seem to forecast the yield and bond price accurately. Percentage error in prediction is less than 1% in both the network models, which indicates a good forecast.

From Figure 2(d) it can be observed that the fit for the parameter λ on the out-of-the-test samples is not quite good. However, the low values for the errors for predicting bond prices using the forecasted parameters suggest that λ does not contribute much toward the forecasting of the yield curve.

Conclusion and Issues

In this chapter, we have successfully established neural networks as a tool for forecasting the term structure of interest rates. The forecasted yield curve can not only be used for predicting the yield and the bond prices but also for predicting various other economic indicators like gross domestic product, growth, inflation, and so on. However, one of the major limitations of the neural network is its inability to explain its behavior. It is difficult, for example, to explain the relative importance of the various inputs. Often, weight-sensitivity analysis is carried out to develop an understanding of the model's behavior. Though we have not done the weight sensitivity analysis, this forms the basis for the formulation of rules, which govern the dynamics of the yield curve as exemplified by the model. Moreover, the generalization ability and hence the predictive ability of the networks decreases as the training time increases. The net is able to decipher the

important features in the dataset after a few passes only. As training progresses, the possibility of overfitting of the data is extremely high. The generated results will have a high R-square with little practical significance, as the ability to recognize and predict patterns outside the training set will be severely hampered. Also, a small change in network design, learning times, and so on, can produce a large change in the network behavior; and, as a result, stability of the predictions by the neural networks may be adversely affected. Choice of an appropriate learning rate is a major issue while developing models based on neural networks. A small value of the learning rate implies lower speed of convergence while a large value of the learning rate results in oscillations. Correctly identifying the optimal network architecture, choosing the appropriate initial weights, and selecting the right activation function governs the predictive capability and hence the performance of the neural network model. It has also been shown that the use of only a few significant variables will produce considerably better results than trying to use every available variable as inputs to the neural network model. Thus, considerable domain expertise is needed while developing a neural network model for various applications. In spite of the improved problem-solving capability of neural networks, tackling the issues cited earlier demands experience on working with the neural networks apart from understanding the domain to which they are being applied. The future scope of work is to develop some sort of rule base using the domain expertise of analysts to make accurate forecasts. This can be achieved by incorporating the fuzzy systems inside the neural network learning.

References

- Adams, K. J., & van Deventer, D. (1994). Fitting yield curves and forward rate curves with maximum smoothness. *Journal of Fixed Income*, 4(1), 52-62.
- Bliss, R. (1997). Testing term structure estimation methods. *Advances in Futures and Options Research*, 9, 97-231.
- Cox, J. C., Ingersoll, J. E., & Ross, S. A. (1985). A theory of the term structure of interest rates. *Econometrica*, 53, 385-407.
- Diebold, F. X., & Li, C. (2002). *Forecasting the term structure of government bond yields*. Unpublished manuscript, University of Pennsylvania, Philadelphia, PA.
- Duffee, G. (2002). Term premia and interest rate forecasts in affine models. *Journal of Finance*, 57, 405-443.
- Duffie, D., & Kan, R. (1996). A yield-factor model of interest rates. *Mathematical Finance*, 6, 379-406.
- Fama, E., & Bliss, R. (1987). The information in long-maturity forward rates. *American Economic Review*, 77, 680-692.
- Fisher, M., Nychka, D., & Zervos, D. (1995). *Fitting the term structure of interest rates with smoothing splines* (FEDS Working Paper).
- Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. Cambridge, MA: MIT Press.

- Haykin, S. (1998). *Neural networks: A comprehensive foundation* (2nd ed.) New York: Prentice-Hall.
- Heath, D., Jarrow, R., & Morton, A. (1992). Bond pricing and the term structure of interest rates: A new methodology for contingent claims valuation. *Econometrica*, 60, 77-105.
- Ho, T. S., & Lee, S. B. (1986). Term structure movements and the pricing of interest rate contingent claims. *Journal of Finance*, 41, 1011-1029.
- Hornik, K. (1993). Some new results on neural network approximation. *Neural Networks*, 6, 1069-1072.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359-366.
- Hull, J., & White, A. (1990). Pricing interest-rate-derivative securities. *Review of Financial Studies*, 3, 573-592.
- Hutchinson, J. M., Lo, A., & Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, 49, 851-889.
- Litterman, R., & Scheinkman, J. (1991). Common factors affecting bond returns. *Journal of Fixed Income*, 1, 51-61.
- McCulloch, J. H. (1971). Measuring the term structure of interest rates. *Journal of Business*, 34, 19-31.
- McCulloch, J. H. (1975). The tax adjusted yield curve. *Journal of Finance*, 30, 811-830.
- McCulloch, J. H., & Kwon, H. (1993). *U.S. term structure data, 1947-1991* (Working Paper 93-6). Ohio State University.
- Mehrotra, K., Mohan, C. K., & Ranka, S. (1997). *Elements of artificial neural networks*. Cambridge, MA: MIT Press.
- Nelson, C. R., & Siegel, A. F. (1987). Parsimonious modeling of yield curves. *Journal of Business*, 60, 473-489.
- Pearson, N., & Sun, T. S. (1994). Exploiting the conditional density in estimating the term structure: An application to the Cox, Ingersoll and Ross model. *Journal of Finance*, 54, 1279-1304.
- Rao, Y. N., & Principe, J. C. (2000). A fast, on-line algorithm for PCA and its convergence characteristics. In *Proceedings IEEE Workshop on Neural Networks for Signal Processing*.
- Refenes, A. N., Azema-Barac, M., Chen, L., & Karoussos, S. A. (1993). *Currency exchange rate prediction and neural network design strategies*. London: Springer-Verlag, Limited.
- Svensson, L. (1994). *Estimating and interpreting forward interest rates: Sweden 1992-1994* (IMF Working Paper No. WP/94/114). Washington, DC: National Bureau of Economic Research, Inc.
- Tanggaard, C. (1997). Nonparametric smoothing of yield curves. *Review of Quantitative Finance and Accounting*, 9(3), 251-267.

- Vasicek, O. A., & Fong, H. G. (1982). Term structure modeling using exponential splines. *Journal of Finance*, 37, 339-348.
- Waggoner, D. (1997). *Spline methods for extracting interest rate curves from coupon bond prices* (Working Paper 97). Georgia: Federal Reserve Bank of Atlanta.
- Whitley, D., & Karunanithi, N. (1991). Generalization in feed forward neural networks. In *International Joint Conference on Neural Networks, Vol. 2*, Seattle, WA (pp. 77-82).

Chapter VIII

Modeling and Prediction of Foreign Currency Exchange Markets

Joarder Kamruzzaman, Monash University, Australia

Ruhul A. Sarker, University of New South Wales, Australia

Rezaul K. Begg, Victoria University, Australia

Abstract

In today's global market economy, currency exchange rates play a vital role in national economy of the trading nations. In this chapter, we present an overview of neural network-based forecasting models for foreign currency exchange (forex) rates. To demonstrate the suitability of neural network in forex forecasting, a case study on the forex rates of six different currencies against the Australian dollar is presented. We used three different learning algorithms in this case study, and a comparison based on several performance metrics and trading profitability is provided. Future research direction for enhancement of neural network models is also discussed.

Introduction

In an era of increasing global competition and integrated economies, the forex rate has become one of the key factors for international trading and open economics. Exchange rates become important when a business or individual purchases goods or services produced in another country because the buyers require to pay the total cost using an appropriate currency demanded by the producer. So the buyers have to purchase other currency for running their businesses. Foreign currency traders make a profit through buying and selling currencies at different rates with fluctuating demands. In fact, the exchange rates play a crucial role in controlling the dynamics of the import-export markets. For example, if the Australian currency is weaker than the U.S. currency, the U.S. traders would prefer to import certain Australian goods, and the Australian producers and traders would find the U.S. as an attractive export market. On the other hand, if Australia is dependent on the U.S. for importing certain goods, it will then be too costly for the Australian consumers under the current exchange rates. In that case, Australia may look for a cheaper source that means shifting from the U.S. to a new import market. As we can imagine, the trade relation and the cost of export/import of goods is directly dependent on the currency exchange rate of the trading partners. Although the foreign exchange market has been estimated at a daily turnover of more than US\$1 trillion (Gan & Ng, 1995), the exchange rates vary continuously during the trading hours. As a result, an accurate prediction of exchange rates is a crucial factor for the success of many businesses and financial institutions.

The risk associated with exchange rate fluctuations that puts companies and individuals into risks has increased substantially over the past decades. In particular, after the breakdown of the Bretton Woods Agreement in the early 1970s, the foreign currency market has become volatile. The market has experienced unprecedented growth over the last few decades, mainly due to floating exchange rates and a push towards further liberalization of trades through the General Agreement on Trade and Tariffs. At times, combined with other financial risks, the exchange rate market becomes so volatile that it contributes to leading the whole national economy into crisis which, for example, was evident in Mexico (1994), Southeast Asia (1997), Russia (1998), and Argentina (2002).

Due to the reasons as outlined earlier, significant efforts have been made over the years to predict foreign exchange rates in order to facilitate financial decision making and risk management. However, exchange rate behavior may exhibit complex characteristics that make it difficult to predict exchange rates within an acceptable accuracy limit (Chinn, 2003). This is illustrated by a recent comment by Alan Greenspan (2002): "There may be more forecasting of exchange rates, with less success, than almost any other economic variable." Furthermore, opposing views existed for years between practicing and academic communities about statistical properties of exchange rates. Practitioners believed exchange rates to have persistent trends while academics considered evidences supporting random walk hypothesis and efficient market hypothesis, which implies that rate changes are independent. The recent empirical studies have presented strong evidence that exchange rates are not independent of the past changes and dismissed the prevalent view in economic literature that exchange rates follow a random walk (Tenti, 1996). There is evidence that shows little support for linear dependence and exhibits the existence of

nonlinearities in exchange rates (Fang, Lai, & Lai, 1994; Grauwe, Dewachter, & Embrechts, 1993). The rates are also characterized as high noise, nonstationary, and chaotic, using high frequency (weekly, daily, or even hourly) past prices (Deboeck, 1994; Tinte, 1996; Yaser & Atiya, 1996). These inherent attributes suggest that past behavior can not be fully exploited to establish the dependency between future rates and that of the past. One general assumption made in such cases is that the historical data incorporate all those behaviors. As a result, the historical data are the major players in the prediction process. Although the well-known conventional forecasting techniques provide predictions for many stable forecasting systems of acceptable quality, these techniques seem inappropriate for non-stationary and chaotic systems such as forex.

Fundamental vs. Technical Analysis

All the various methods that have been developed in modeling forex rates can be broadly categorized into the following two groups.

- *Fundamental analysis* — In this case, the analysis is based on the exact knowledge of various factors that influence the economy and the relationship between those factors. The analysis focuses in depth at the financial condition of the country and studies the effect of supply and demand on each currency. The empirical models, like, the balance-of-payment-flow model, currency substitution model, monetary model of forex, and byrid monetary/fiscal policy model are some of the examples of fundamental analysis.
- *Technical analysis* — In this case, the prediction relies on the discovery of strong empirical regularities by analyzing a set of historical data by various methods like time series analysis, regression analysis, expert systems, and so on. It assumes that the future movement follows some trends and these trends can be captured.

The main problem with fundamental analysis is that the knowledge of the rules that govern the forex behavior is not readily available (Kodogiannis & Lolis, 2001). Research has shown that fundamental analysis-based models can be used to explore the long-term trends in forex movements but are inadequate in explaining the short- and medium-term fluctuations (Rosenberg, 1981). On the other hand, many technical-analysis-based models have been found to be successful in forecasting short-term exchange rates. The success of technical analysis-based models has made this analysis extremely popular among market participants (Ghoshray, 1996).

Neural Network-Based Forecasting of Exchange Rates

Many techniques have been proposed over the past few decades for reliable forecasting of exchange rates. The traditional statistical forecasting methods — for example, the Box-Jenkins' Auto-Regressive Integrated Moving Average (ARIMA) — have relied on linear models (Box & Jenkins, 1976). However, ARIMA is a general univariate model and it is developed based on the assumption that the time series being forecasted are linear and stationary. The drawback of the linear model has led to the development of alternative methods among which artificial neural networks (ANNs) have emerged as a promising forecasting tool. Over the last decade, researchers and practitioners alike have shown growing interest in applying ANNs in time series analysis and forecasting. ANNs are an effective tool to realize any nonlinear input-output mapping. It has been demonstrated that, with sufficient number of hidden layer units, an ANN is capable of approximating any continuous function to any desired degree of accuracy (Cybenko, 1989). Due to the nature of their learning process, ANNs can be regarded as nonlinear autoregressive models.

Neural networks, well known for their approximation capability in prediction and system modeling, have recently demonstrated their great applicability in many time series analysis and forecasting applications in diverse disciplines. ANNs assist multivariate analysis. Multivariate models can rely on greater information, where not only the lagged time series is being forecast, but also other indicators (such as technical, fundamental, intermarker, etc., for the financial market) are combined to act as predictors. In addition, ANNs are more effective in describing the dynamics of nonstationary time series due to their unique nonparametric, noise-tolerant, and adaptive properties.

One of the early works using ANNs for forex forecasting was done by Refenes, Barac, Chen, and Karoussos (1992). The system used a standard backpropagation algorithm (Rumelhart, 1986) to predict the exchange rate between the U.S. dollar and deutsche mark using the data for the period 1988-1989 on hourly updates. The architecture consisted of a two-layer network with a fixed number of inputs modeling a window moving along the time in fixed steps. The first 6 months were used for training and the following 6 months as the test set. The network produced accurate predictions, making at least 20% profit on the last 60 trading days of 1989. Gan and Ng (1995) developed two ANN models, also built on standard backpropagation algorithm, using univariate and multivariate time series to forecast the Swiss franc, deutsche mark, and yen against the U.S. dollar. Their results showed that ANN models were able to predict better than a benchmark random walk model. Tenti (1996) proposed recurrent neural network models with the view that such networks, where input layers' activity patterns pass through the network more than once before generating a new output pattern, are capable of learning extremely complex patterns. He tested three different recurrent architectures by comparing their prediction accuracy of the deutsche mark against the U.S. dollar. The author reported that two of three recurrent architectures produced better profitability and generalization ability than standard backpropagation on repeated tests performed in his experiments. Profitability was measured in terms of Return on Equity (ROE) and Return on Capital (ROC).

Francesco and Schiavo (1999) performed experiments with a long period of data, the monthly exchange rate of four major European currencies (French franc, deutsche mark, Italian lira and British pound against the U.S. dollar) from 1973 to 1995 to provide a comparative evaluation of neural network and chaotic models over the same data sets and variables. A two-layer network trained by standard backpropagation was considered. The prediction performances were measured in terms of normalized mean-squared error and statistical significance. Neural networks performed better than chaotic models, and both models performed substantially better than random walk. In terms of statistical significance by Mizrach's test, both models were found to be statistically equivalent.

Yao and Tan (2000) used technical indicators and time series data to build a neural network model using weekly data for the period of May 1984 to October 1993 of Singapore Foreign Exchange and predicted closing prices for the period of November 1993 to July 1995. They also used a two-layer network trained by a standard backpropagation algorithm. A validation set was used to build the model. Six major currencies were studied. It was shown that a network built on a time delayed series can only predict rate movement direction by little above 50%, while it rises to about 74% when the model was built using technical indicators. The results also confirmed that such a neural network model performed significantly better than the traditional ARIMA model when compared in terms of normalized mean-squared error, directional change of movement, and profitability. The works showed that, without extensive market data or knowledge, useful prediction and significant profit could be made by a neural network-based forex forecasting model (Yao & Tan, 2000). Extensive experimentation with a single currency (Swiss franc) established the consistency of a neural network's ability in forecasting exchange rates. A similar study by Voginovic, Kecman, and Seidel (2001) used a radial basis function neural network for forecasting the daily closing exchange rate of the New Zealand dollar against the U.S. dollar. The model performed significantly better than traditional linear autoregressive model in both directional change and accuracy. The study also investigated the impact of model order, number of hidden layer and training set size on prediction accuracy.

The neural network models for forex prediction are usually trained off-line. MacDonald (1999) and Schinasi et al. (1989) argued that a more appropriate way of enhancing performance of prediction models would be to allow the coefficients to evolve over time. Economic theories also indicate that factors like money demand instabilities, policy changes, and global trade patterns could lead to parameter instability and a changed relationship. Minghui, Sratchandran, and Sundararajan (2003) proposed an online training of a neural network called Minimum Resource Allocating Network (MRAN) to address the issue of time varying parameters. The MRAN model incorporating economic fundamentals as inputs was found to perform much better than random walk and feed-forward neural network models. The study claimed that the MRAN model could forecast the trend turning points accurately for some periods, while the random walk model was incapable of doing so. Chen and Leung (2004) proposed a hybrid approach to model building that employed two stages. First, a time series model generates estimates of exchange rates, then a general regression neural network (details provided in Chapter I of this book) corrects the errors of the estimates. Chen and Leung used a data set provided by the International Monetary Fund that covered 22 years from January 1980 to December 2001 and adopted the macroeconomic variables expressed in modified Uncovered

Interest Parity (UIP) relationship. Results showed the two-stage hybrid approach produced more accurate exchange rate prediction and higher return (0.61% improvement in annual return) than a single stage model.

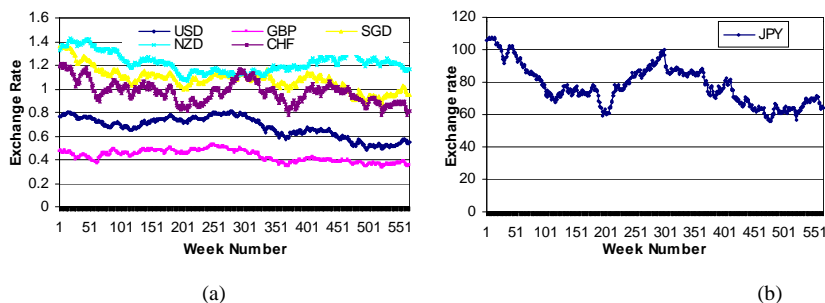
Apart from simply using ANN models for prediction, several other studies used neural networks to enhance the understanding of empirical phenomena. White and Racine (2001) used the bootstrap method of inference using neural networks to test a forex market efficiency hypothesis. They concluded that there is evidence that supports the existence of predictive information in past rate changes, but “the nature of the predictive relations evolves through time.”

A Case Study: Australian Forex Market

In the following, we present a case study to forecast six different currency rates, namely, the U.S. dollar (USD), Great British pound (GBP), Japanese yen (JPY), Singapore dollar (SGD), New Zealand dollar (NZD) and Swiss franc (CHF) against the Australian dollar using their historical exchange rate data. The case study is based on our previous study (Kamruzzaman & Sarker, 2004). In most of the previous studies related to forex forecasting, the neural network algorithms used were: Standard Backpropagation (SBP), Radial Basis Function (RBF), or Generalized Regression Neural Network (GRNN). In this case study, we used two other improved feed-forward learning algorithms, namely the Scaled Conjugated Gradient (SCG) and Bayesian Regularization (BR) algorithms, to build the model and investigate how the algorithms performed compared to standard backpropagation in terms of prediction accuracy and profitability.

Dataset. The data used in this study is the foreign exchange rate of six different currencies against the Australian dollar from January 1991 to July 2002 made available by the Reserve Bank of Australia. A total of 565 weekly data of the previously mentioned currencies were considered, of which first 500 weekly data were used for training and the remaining 65 weekly data for evaluating the model. The plots of historical rates for USD, GBP, SGD, NZD, and CHF are shown in Figure 1(a) and for JPY in Figure 1(b).

Figure 1. Historical exchange rates for (a) USD, GBP, SGD, NZD and CHF and (b) JPN against Australian dollar



Technical Indicators. The time delay moving average is used as a technical indicator. The advantage of the moving average is its tendency to smooth out some of the irregularity that exists between market days. We used moving average values of past weeks to feed to the neural network to predict the following week's rate. The indicators are MA5, MA10, MA20, MA60, MA120, and X_t , namely, moving average of 1 week, 2 weeks, 1 month, 1 quarter, half year, and last week's closing rate, respectively. The predicted value is X_{t+1} . The model has six inputs for six indicators, one hidden layer, and one output unit to predict exchange rate. It has been reported in another study that increasing the number of inputs does not necessarily improve forecasting performance (Yao & Tan, 2000).

Learning Algorithms. In most of the previous studies, a standard backpropagation algorithm has been investigated. However, backpropagation suffers from slow convergence and sometimes fails to learn the time series within a reasonable computational time limit. A desired neural network model should produce small error not only on sample data but also on out of sample data. In this case study, we investigated with Scaled Conjugated Gradient (Moller, 1993) and Bayesian Regularization (MacKey, 1992) algorithms that have been reported to produce improved results than the standard backpropagation in a number of other studies. A detailed description of the algorithms is presented in Chapter 1 of this book.

Evaluation of Prediction Accuracy. The most common measure to evaluate how closely the model is capable of predicting future rate is measured by Normalized Mean-Square Error (NMSE). The other measure important to the trader is correct prediction of movement. We used four other measures, which are: Mean Absolute Error (MAE), Directional Symmetry (DS), Correct Up trend (CU) and Correct Down trend (CD). These criteria are defined in Table 1, where x_k and \hat{x}_k are the actual and predicted values,

Table 1. Performance metrics to evaluate the forecasting accuracy of the model

$NMSE = \frac{\sum_k (x_k - \hat{x}_k)^2}{\sum_k (x_k - \bar{x}_k)^2} = \frac{1}{\sigma^2 N} \sum_k (x_k - \hat{x}_k)^2$
$MAE = \frac{1}{N} x_k - \hat{x}_k $
$DS = \frac{100}{N} \sum_k d_k, \quad d_k = \begin{cases} 1 & \text{if } (x_k - x_{k-1})(\hat{x}_k - \hat{x}_{k-1}) \geq 0 \\ 0 & \text{otherwise} \end{cases}$
$CU = 100 \frac{\sum_k d_k}{\sum_k t_k},$
$d_k = \begin{cases} 1 & \text{if } (\hat{x}_k - \hat{x}_{k-1}) > 0, (x_k - x_{k-1})(\hat{x}_k - \hat{x}_{k-1}) \geq 0 \\ 0 & \text{otherwise} \end{cases}, \quad t_k = \begin{cases} 1 & \text{if } (x_k - x_{k-1}) > 0 \\ 0 & \text{otherwise} \end{cases}$
$CD = 100 \frac{\sum_k d_k}{\sum_k t_k}$
$d_k = \begin{cases} 1 & \text{if } (\hat{x}_k - \hat{x}_{k-1}) < 0, (x_k - x_{k-1})(\hat{x}_k - \hat{x}_{k-1}) \geq 0 \\ 0 & \text{otherwise} \end{cases}, \quad t_k = \begin{cases} 1 & \text{if } (x_k - x_{k-1}) < 0 \\ 0 & \text{otherwise} \end{cases}$

respectively. NMSE and MAE measure the deviation between actual and forecast value. Smaller values of these metrics indicate higher accuracy in forecasting. Additional evaluation measures include the calculation of correct matching number of the actual and predicted values with respect to sign and directional change. DS measures correctness in predicted directions while CU and CD measure the correctness of predicted up and down trends, respectively.

Profitability. The traders are more interested in making a profit by buying and selling in forex market. In order to assess the profitability attainable by using the model, we simulated a trading over the forecasted period. Similar simulated trading is also used in another study (Yao & Tan, 2000). Seed money is used to trade according to the following strategy:

if $(\hat{x}_{k+1} - x_k) > 0$ then buy otherwise sell.

At the ending the trading period, the currency is converted to the original seed money. The profit return is then calculated as:

$$Return = \left(\frac{Money\ Obtained}{Seed\ Money} \right)^{52/w} - 1$$

where *Money Obtained* is the money at the end of the testing period and w is the number of weeks in the testing period.

Simulation Results

Neural network models were trained with six inputs representing the six technical indicators, a hidden layer, and an output unit to predict the exchange rate. Since the final model at the end of training a neural network depends on many factors, like, number of hidden units, parameter setting, initial weights, stopping criteria, and so on, we trained 30 different networks with different initial weights, learning parameters, and hidden unit number. The number of hidden units was varied between 3 to 7 and the training was terminated at iteration number between 5000 to 10000. Out of all the trials, the network that yielded the best result in each algorithm is presented here.

We measured the performance metrics on the test data to investigate how well the neural network forecasting model captured the underlying trend of the movement of each currency against Australian dollar. Table 2 shows the performance metrics achieved by each model over a forecasting period of 35 weeks and Table 3 shows the same over 65 weeks (previous 35 weeks plus additional 30 weeks) and also compared with an ARIMA-based model. All the ANN-based models perform much superiorly compared to the ARIMA model in respect to all the evaluation criteria. This finding is consistent with other studies (Yao & Tan, 2000; Zoran et al. 2001). The results show that the SCG and BR models consistently perform better than the SBP model in terms of all performance metrics in almost all the currency exchange rates. For example, in case of forecasting the

U.S. dollar rate over 35 weeks, the NMSE achieved by SCG and BR is quite low and is almost half of that achieved by SBP. This means these models are capable of predicting exchange rates more closely than SBP. Also, in predicting trend directions SCG and BR is almost 10% more accurate than SBP. The reason of better performance by the SCG and BR algorithms is the improved learning technique, which allows them to search efficiently in weight space for solution. Similar trend is observed in predicting other currencies.

Between the SCG and BR models, the former performs better in all currencies except the Japanese yen in terms of the two most commonly used criteria, that is, NMSE and MAE. In terms of other metrics, SCG yields slightly better performance in the case of the Swiss franc, the BR was slightly better in the U.S. dollar and British pound, and both the SCG and BR perform equally in case of the Japanese yen and the Singapore and New Zealand dollars. In both algorithms, the directional change prediction accuracy is above 80%, which is much improved from the 70% accuracy achieved in a similar study (Yao & Tan, 2000).

In building a neural network model, we need to be attentive to few factors that influence the performance of the network. The generalization ability of neural networks, that is, its ability to produce correct output in response to an unseen input is influenced by a number

Table 2. Measurement of prediction performance over 35-week prediction

Currency	NN model	Performance metrics				
		NMSE	MAE	DS	CU	CD
U.S. dollar	ARIMA	1.0322	0.0069	52.94	0.00	100.00
	SBP	0.5041	0.0047	71.42	76.47	70.58
	SCG	0.2366	0.0033	82.85	82.35	88.23
	BR	0.2787	0.0036	82.85	82.35	88.23
B. pound	ARIMA	0.9344	0.0065	55.88	0.00	100.00
	SBP	0.5388	0.0053	77.14	75.00	78.94
	SCG	0.1578	0.0030	77.14	81.25	73.68
	BR	0.1724	0.0031	82.85	93.75	73.68
J. yen	ARIMA	1.2220	1.77859	38.23	0.00	100.00
	SBP	0.1530	0.6372	74.28	72.72	76.92
	SCG	0.1264	0.6243	80.00	81.81	76.92
	BR	0.1091	0.5806	80.00	81.81	76.92
S. dollar	ARIMA	1.1765	0.0184	52.94	0.00	100.00
	SBP	0.2950	0.0094	85.71	82.35	88.88
	SCG	0.2321	0.0076	82.85	82.35	83.33
	BR	0.2495	0.0080	82.85	82.35	83.33
NZ dollar	ARIMA	0.9728	0.0139	52.94	0.00	100.00
	SBP	0.1200	0.0046	77.14	75.00	78.94
	SCG	0.0878	0.0038	85.71	87.50	84.21
	BR	0.0898	0.0039	85.71	87.50	84.21
S. franc	ARIMA	0.9378	0.0285	44.11	0.00	100.00
	SBP	0.1316	0.0101	80.00	75.00	86.66
	SCG	0.0485	0.0059	82.85	80.00	86.66
	BR	0.0496	0.0057	80.00	75.00	86.66

of factors: (1) the size of the training set, (2) the degrees of freedom of the network related to the architecture, and (3) the physical complexity of the problem at hand. Practically, we have no control on the problem complexity, and in our simulation the size of the training set is fixed. This leaves the generalization ability, that is, the performance of the model dependent on the architecture of the corresponding neural network. Generalization performance can also be related to the complexity of the model in the sense that, in order to achieve best generalization, it is important to optimize the complexity of the prediction model (Bishop, 1995). In the case of neural networks, changing the number of adaptive parameters in the network can vary the complexity. A network with fewer weights is less complex than the one with more weights. It is well known that the “simplest hypothesis/model is least likely to overfit.” A network that uses the least number of weights and biases to achieve a given mapping is least likely to overfit the data and is most likely to generalize well on the unseen data. If redundancy is added in the form of extra hidden unit or additional parameters, it is likely to degrade performance because more than the necessary number of parameters is used to achieve the same mapping. In the case of nonlinear regression, two extreme solutions should be avoided: filtering out the underlying function or underfitting (not enough hidden neurons), or modeling of noise or overfitting data (too many hidden neurons). This situation is also known as bias-

Table 3. Measurement of prediction performance over a 65-week prediction

Currency	NN model	Performance metrics				
		NMSE	MAE	DS	CU	CD
U.S. dollar	ARIMA	1.7187	0.0171	42.19	0.00	100.00
	SBP	0.0937	0.0043	75.38	81.57	69.23
	SCG	0.0437	0.0031	83.07	78.94	92.30
	BR	0.0441	0.0030	83.07	78.94	92.30
B. pound	ARIMA	1.2791	0.0094	50.00	0.00	100.00
	SBP	0.2231	0.0038	80.00	75.75	87.09
	SCG	0.0729	0.0023	84.61	87.87	83.87
	BR	0.0790	0.0024	87.69	93.93	83.87
J. yen	ARIMA	2.3872	4.1329	43.75	0.00	100.00
	SBP	0.0502	0.5603	76.92	75.67	78.57
	SCG	0.0411	0.5188	81.53	83.78	78.57
	BR	0.0367	0.5043	81.53	83.78	78.57
S. dollar	ARIMA	1.6472	0.0313	48.43	0.00	100.00
	SBP	0.0935	0.0069	83.07	82.35	83.87
	SCG	0.0760	0.0060	86.15	88.23	83.87
	BR	0.0827	0.0063	86.15	88.23	83.87
NZ dollar	ARIMA	1.1365	0.0233	56.25	0.00	100.00
	SBP	0.0342	0.0042	78.46	71.42	86.11
	SCG	0.0217	0.0033	84.61	82.14	88.88
	BR	0.0221	0.0033	84.61	82.14	88.88
S. franc	ARIMA	0.9158	0.0273	46.87	0.00	100.00
	SBP	0.1266	0.0098	83.07	80.00	86.66
	SCG	0.0389	0.0052	84.61	84.61	86.66
	BR	0.0413	0.0051	81.53	77.14	86.66

Table 4. Return of currency trading of different models

U.S. dollar	9.29	14.17	14.20
B. pound	10.16	15.81	12.21
J. yen	6.29	10.91	11.58
S. dollar	9.07	7.56	8.41
NZ dollar	6.48	8.73	9.02
S. franc	3.23	4.15	1.98

variance dilemma. One way of controlling the effective complexity of the model in practice is to compare a range of models having different architectures and to select the one that yields the best performance on the test data.

Although the performance measures presented earlier show the better performance of SCG and BR models over a standard backpropagation model, the profitability of these models plays an important roll in the actual trading environment. Table 4 shows the profitability attainable by various models over the testing period. This confirms the better performance of SCG and BR over standard backpropagation in terms of trading prospect.

Conclusion

In this chapter, we have presented an overview of foreign exchange rate prediction, especially by models based on neural networks, and presented a case study on Australian foreign exchange market. Neural network models are well suited to learn nonlinearities in the exchange rates, the existence of which is evidenced in many studies. However, the performance of such models depends on the economic variables used in modeling, as well as how well a particular learning algorithm can generalize on the sample data. In the case study, we investigated with three different neural network learning algorithms and found that scaled conjugate gradient and Bayesian regularization were significantly better in performance than standard backpropagation. Though a few other algorithms have been studied in the literature, there is a need for a more comprehensive study with extensive data to determine which algorithm is best suited for modeling exchange rate prediction, both short- and long-term. For example, genetic algorithms may be used to select the optimum architecture and parameters of a neural network during the training phase or a hybrid system using fuzzy measure and wavelet techniques may lead to a better prediction.

References

- Bishop, C. M. (1995). *Neural networks for pattern recognition*. New York: Oxford University Press.
- Box, G. E. P., & Jenkins, G. M. (1990). *Time series analysis: Forecasting and control*. San Francisco: Holden-Day.
- Chen, A. S., & Leung, M. T. (2004). Regression neural network for error correction in foreign exchange forecasting and trading. *Computers and Operations Research*, *31*, 1049-1068.
- Chinn, M. (2003). *Explaining exchange rate behavior*. National Bureau of Economic Research.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303-314.
- Deboeck, G. (1994). *Trading on the edge: Neural, genetic and fuzzy systems for chaotic financial markets*. New York: Wiley.
- Fang, H., Lai, S., & Lai, M. (1994). Fractal structure in currency futures price dynamics. *Journal of Futures Markets*, *14*, 169-181.
- Francesco, L., & Schiavo, R. A. (1999). A comparison between neural networks and chaotic models for exchange rate prediction. *Computational Statistics & Data Analysis*, *30*, 87-102.
- Gan, W. S., & Ng, K. H. (1995). Multivariate FOREX forecasting using artificial neural networks. In *Proceedings IEEE International Conference on Neural Networks*, Vol. 2 (pp. 1018-1022).
- Ghoshray, S. (1996). Currency exchange rate prediction technique by fuzzy inferencing on the chaotic nature of time series data. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *4*(5), 431-448.
- Grauwe, D. P., Dewachter, H., & Embrechts, M. (1993). *Exchange rate theory: chaotic models of foreign exchange markets*. London: Blackwell.
- Greenspan, A. (2002, June 16). *Testimony of the federal reserve board's semiannual monetary policy report to the congress, before the Committee on Banking, Housing, and Urban Affairs, U.S. Senate*. Washington, D.C.
- Kamruzzaman, J., & Sarker, R. (2004). ANN-based forecasting of foreign currency exchange rates. *Neural Information Processing — Letter & Review*, *3*(2), 49-58.
- Kodogiannis, V., & Lolis, A. (2001). A comparison between neural network and fuzzy system models for foreign exchange rates prediction. *Neural, Parallel, & Scientific Computations*, *9*(3-4), 417-427.
- MacDonald, R. (1999). What do we really know about real exchange rates? In R. MacDonald, & J. Stein (Eds.), *Equilibrium exchange rates*. Amsterdam: Kluwer.
- Mackay, D. J. C. (1992). Bayesian interpolation. *Neural Computation*, *4*, 415-447.

- Medeiros, M. C., Veiga, A., & Pedreira, C. E. (2001). Modeling exchange rates: Smooth transitions, neural network and linear models. *IEEE Transaction Neural Networks*, 12(4), 2001.
- Minghui, H., Sratchandran, P., & Sundararajan, N. (2003). A sequential learning neural network for foreign exchange rate forecasting. *IEEE International Conference on Systems, Man and Cybernetics*, 4, 3963-3968.
- Moller, A. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6, 525-533.
- Refenes, A. N., Barac, M. A., Chen, L., & Karoussos, A. S. (1992). Currency exchange rate prediction and neural network design strategies. *Neural Computing and Applications*, 1, 46-58.
- Rosenberg, M. (1981, June). Is technical analysis right for currency forecasting? *Euromoney*, 125-131.
- Rumelhart, D. E., McClelland, J. L., & the PDP Research Group (1986). *Parallel Distributed Processing, 1*. MIT Press.
- Schinasi, G., & Swamy, P. A. (1989). The out-of-sample forecasting performance of exchange rate models when coefficients are allowed to change. *Journal of International Money and Finance*, 8, 375-390.
- Tenti, P. (1996). Forecasting foreign exchange rates using recurrent neural networks. *Applied Artificial Intelligence*, 10, 567-581.
- Vojinovic, Z., Kecman, V., & Seidel, R. (2001). A data mining approach to financial time series modeling and forecasting. *International Journal of Intelligent Systems in Accounting, Finance & Management*, 10, 225-239.
- Yao, J., Li, Y., & Tan, C. L. (2000). Option price forecasting using neural networks. *OMEGA: International Journal of Management Science*, 28, 455-466.
- Yao, J., & Tan, C. T. (2000). A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, 34, 79-98.
- Yaser, S., & Atiya, A. (1996). Introduction to Financial Forecasting, *Applied Intelligence*, 6, 205-213.
- Zhang, G., & Hu, M. Y. (1998). Neural network forecasting of the British Pound/US dollar exchange rate. *OMEGA: International Journal of Management Science*, 26, 495-506.
- Zhenyuan, W., Yilu, L., & Griffin, P. J. (2000). Neural net and expert system diagnose transformer faults. *Computer Applications in Power*, 13(1), 50-55.

Chapter IX

Improving Returns On Stock Investment through Neural Network Selection

Tong-Seng Quah, Nanyang Technological University, Republic of Singapore

Abstract

Artificial neural networks' (ANNs') generalization powers have in recent years received admiration of finance researchers and practitioners. Their usage in such areas as bankruptcy prediction, debt-risk assessment, and security-market applications has yielded promising results. With such intensive research and proven ability of the ANN in the area of security-market application and the growing importance of the role of equity securities in Singapore, it has motivated the conceptual development of this work in using the ANN in stock selection. With their proven generalization ability, neural networks are able to infer the characteristics of performing stocks from the historical patterns. The performance of stocks is reflective of the profitability and quality of management of the underlying company. Such information is reflected in financial and technical variables. As such, the ANN is used as a tool to uncover the intricate relationships between the performance of stocks and the related financial and technical variables. Historical data, such as financial variables (inputs) and performance of the stock (output) is used in this ANN application. Experimental results obtained thus far have been very encouraging.

Introduction

With the growing importance in the role of equities to both international and local investors, the selection of attractive stocks is of utmost importance to ensure a good return. Therefore, a reliable tool in the selection process can be of great assistance to these investors. An effective and efficient tool/system gives the investor the competitive edge over others as he/she can identify the performing stocks with minimum effort.

In assisting the investors in their decision-making process, both the academics and practitioners have devised trading strategies, rules, and concepts based on fundamental and technical analysis. Innovative investors opt to employ information technology to improve the efficiency in the process. This is done through transforming trading strategies into computer-known language so as to exploit the logical processing power of the computer. This greatly reduces the time and effort in short-listing the list of attractive stocks.

In the age where information technology is dominant, such computerized rule-based expert systems have severe limitations that will affect their effectiveness and efficiency. In particular, their inability in handling nonlinear relationships between financial variables and stock prices has been a major shortcoming. However, with the significant advancement in the field of ANNs, these limitations have found a solution. In this work, the generalization ability of the ANN is being harnessed in creating an effective and efficient tool for stock selection. Results of the research in this field have so far been very encouraging.

Application of Neural Network in Stock Investment

One of the earliest studies was by Halquist and Schmoll (1989), who used a neural network model to predict trends in the S&P 500 index. They found that the model was able to predict the trends 61% of the time. This was followed by Trippi and DeSieno (1992) and Grudnitski and Osburn (1993). Trippi and DeSieno (1992) devised an S&P 500 trading system that consisted of several trained neural networks and a set of rules for combining the network results to generate a composite recommended trading strategy. The trading system was used to predict S&P 500 index futures and the results showed that this system significantly outperformed the passive buy-and-hold strategy. Grudnitski and Osburn (1993) used a neural network to predict the monthly price changes and trading return in the S&P 500 index futures. The results showed that the neural network was able to predict correctly 75% of the time and gave a positive return above risk.

Another work on predicting S&P 500 index futures was by Tsaih, Hsu, and Lai (1998). Similar to Trippi and DeSieno (1992), Tsaih et al. (1998) also integrated a rule-based system technique with a neural network to produce a trading system. However, in the Tsaih et al. (1998) study, they used reasoning neural networks instead of the backpropagation method used by Trippi and Desieno (1992). Empirical results in the daily prediction of price changes in the S&P 500 index futures showed that this hybrid artificial-intelligence (AI) approach outperformed the passive buy-and-hold investment strategy.

Similar works on predicting the S&P 500 index were also carried out by Min (1999) and Min and Maddala (1999). Essentially, these two papers were trying to compare whether nonlinear models like neural networks were able to show better predictive accuracy than linear models like the linear regression, since several studies had shown the nonlinearity in stock returns. Both papers had shown that the predictive accuracy of stock returns by neural network models was better than their linear counterparts both in in-sample and out-of-sample forecasts.

Encouraged by the success of earlier researchers, many now apply neural networks in modeling their national stock markets. Olson and Mossman (2003) forecasted the Canadian stock returns by training a neural network to recognize relationships between accounting ratios and stock price movements. Likewise, Chen, Leung, and Daouk (2003) used an ANN to forecast and trade on the Taiwan stock index. Perez, Torra, and Andrada (2005) found that ANNs consistently outperformed auto-regression models in forecasting the Spanish Ibex-35 stock index. Last but not least, Cao, Leggio, and Schniederjans (2005) applied ANNs on the Chinese stock market and obtained results that are better than linear models.

ANN Model for Stock Selection

In this section, the architecture and design of the ANN model are described.

Neural Architecture

The computer software selected for training and testing the network is Neural Planner version 3.71. Stephen Wolstenholme programmed this software. It is an ANN simulator strictly designed for only one backpropagation learning algorithm. There are four major issues in the selection of the appropriate network (Gately, 1996):

1. Selection of the appropriate algorithm.
2. Architecture of the ANN.
3. Selection of the learning rule.
4. Selection of the appropriate learning rates and momentum.

Select the Appropriate Algorithm

The sole purpose of this work is to identify the top performing stocks, and the historical data that is used for the training process will have a known outcome (whether it is considered top performer or otherwise). Therefore, algorithms designed for supervised learning are ideal. Among the available algorithms, the backpropagation algorithm designed by Rumelhart, Hinton, and Williams (1986) is the most suitable, as it is being

intensively tested in finance. Moreover, it is recognized as a good algorithm for generalization purposes.

Architecture of ANN

Architecture, in this context, refers to the entire structural design of the ANN, including the input layer, hidden layer, and output layer. It involves determining the appropriate number of neurons required for each layer and also the appropriate number of layers within the hidden layer. The logic of the backpropagation method is the hidden layer. The hidden layer can be considered as the crux of the backpropagation method. This is because the hidden layer can extract higher-level features and facilitate generalization, if the input vectors have low-level features of a problem domain or if the output/input relationship is complex. The fewer the hidden units, the better is the ANN able to generalize. It is important not to overfit the ANN with a larger number of hidden units than required until it can memorize the data. This is because the nature of the hidden units is like a storage device. It learns noise present in the training set, as well as the key structures. No generalization ability can be expected in these. This is undesirable, as it does not have much explanatory power in a different situation/environment.

Selection of the Learning Rule

The learning rule is the rule that the network follows in its error-reducing process. This is to facilitate the derivation of the relationships between the input(s) and output(s). The generalized delta rule developed by Rumelhart, et al. (1986) is used in the calculations of the weights. This particular rule is selected because it is widely used and proven effective in finance research.

Selection of the Appropriate Learning Rate and Momentum

The learning rate and momentum are parameters in the learning rule that aid the convergence of error, so as to arrive at the appropriate weights that are representative of the existing relationships between the input(s) and the output(s).

As for the appropriate learning rate and momentum to use, the NEURAL PLANNER Software has a feature that can determine appropriate learning rate and momentum with which the network will be able to start training. This function is known as “Smart Start.” Once this function is activated, the network will be tested using different values of learning rate and momentum to find a combination that yields the lowest average error after a single learning cycle. These are the optimum starting values, as using these rates improves the error-converging process thus requiring less processing time.

Another attractive feature is that the software comes with an “auto-decay” function that can be enabled or disabled. This function automatically adjusts the learning rate and momentum to enable a faster and more accurate convergence. In this function, the software will sample the average error periodically, and if it is higher than the previous

sample then the learning rate is reduced by 1%. The momentum is “decayed” using the same method but the sampling rate is half of that used for the learning rate. If both the learning rate and momentum decay are enabled, then the momentum will decay slower than the learning rate.

In general cases, where these features are not available, a high learning rate and momentum (e.g., 0.9 for both the learning rate and momentum) are recommended as the network will converge at a faster rate than when lower figures are used. However, too high a learning rate and momentum will cause the error to oscillate and thus prevent the converging process. Therefore, the choice of learning rate and momentum are dependent on the structure of the data and the objective of using the ANN.

Variables Selection

In general, financial variables chosen are constrained by data availability. They are chosen first on the significant influences over stock returns based on past literature searches and practitioners’ opinions and then on the availability of such data. Most data used in this research is provided by Credit Lyonnais Securities (2005). Stock prices are extracted from Bloomberg (2005) financial database.

Broadly, factors that can affect stocks prices can be classified into three categories: economic factors, political factors, and firm/stock specific factors. Economic factors have significant influence on the returns of individual stock as well as stock index in general as they possess significant impact on the growth and earnings’ prospects of the underlying companies thus affecting the valuation and returns. Moreover, economic variables also have significant influence on the liquidity of the stock market. Some of the economic variables used are: inflation rates, employment figures, and producers’ price index.

Many researchers have found that it is difficult to account for more than one third of the monthly variations in individual stock returns on the basis of systematic economic influences and shown that political factors could help to explain some of the missing variations. Political stability is vital to the existence of business activities and the main driving force in building a strong and stable economy. Therefore, it is only natural that political factors such as fiscal policies, budget surplus/deficit, and so on do have effects on stock price movements.

Firm specific factors affect only individual stock returns. For example, financial ratios and some technical information that affects the return structure of specific stocks, such as yield factors, growth factors, momentum factors, risk factors, and liquidity factors. As far as stock selection is concerned, firm specific factors constitute to important considerations, as it is these factors that determine whether a firm is a bright star or a dim light in the industry. Such firm specific factors can be classified into five major categories:

1. Yield factors: These include “historical P/E ratio” and “prospective P/E ratio.” The former is computed by price/earning per share; the latter is derived by price/consensus earnings per share estimate. Another variable is the “cashflow yield,” which is basically price/operating cashflow of the latest 12 months.

2. Liquidity factors: The most important variable is the “market capitalization,” which is determined by “price of share x number of shares outstanding.”
3. Risk factors: The representative variable is the “earning per share uncertainty,” which is defined as “percentage deviation about the median Earning Per Share (EPS) estimates.”
4. Growth factors: Basically, this means the “return on equity (ROE),” and is computed by “net profit after tax before extraordinary items/shareholders equity.”
5. Momentum factors: A proxy is derived by “average of the price appreciation over the quarter with half of its weights on the last month and remaining weights being distributed equally in the remaining two months.”

The inputs of the neural network stock selections system are the previous seven inputs and the output is the return differences between the stock and the market return (excess returns). This is to enable the neural network to establish the relationships between inputs and the output (excess returns). In this work, political factors and economic factors are not taken into consideration as the stock counters used in this study are listed on the same stock exchange (and are therefore subjected to the same forces).

The training data set includes all data available until the quarter before the testing quarter. This is to ensure that the latest changes in the relationship of the inputs and the output are being captured in the training process.

Experiment

The quarterly data required in this work is generally stock prices and financial variables (inputs to the ANN stock selection system) from 1/1/93 to 31/12/96. Credit Lyonnais Securities (Singapore) Pte Ltd. provided most of the data used in this work.

The download stock prices served as the basis to calculate stock returns. These stock returns — adjusted for dividends, stock splits, and bonus issues — will be used as output in the ANN-training process.

One unique feature of this research is that the prospective P/E ratio, measured as price/consensus earnings per share estimate, is being used as a forecasting variable. This variable has not received much attention in financial research. Prospective P/E ratio is used among practitioners as it can reflect the perceived value of stock with respect to earnings-per-share (EPS) expectations. It is used as a value indicator, which has similar implications as that of the historical P/E ratio. As such, a low prospective P/E suggests that the stock is undervalued with respect to its future earnings and vice versa. With its explanatory power, prospective P/E ratio qualifies as an input in the stock selection system. Data on earnings-per-share estimates, which is used for the calculation of EPS uncertainty and prospective P/E ratio, is available in the Estimates Directory (Singapore Exchange, 2006). This is a compilation of EPS estimates and recommendations put forward by financial analysts. The coverage has estimates from January 1993.

Research Design

The purpose of this ANN stock selection system is to select stocks that are top performers from the market (stock that outperformed the market by 5%) and to avoid selecting under performers (stocks that under-performed the market by 5%). More importantly, the aim is to beat the market benchmark (quarterly return on the SESALL index) on a portfolio basis.

This ANN stock selection system employs a quarterly portfolio rebalancing strategy whereby it will select stocks in the beginning of the quarter, and performance (the return of the portfolio) will be assessed at the end of the quarter.

Design 1 (Basic System)

In this research design, the sample used for training consists of stocks that out- and underperformed the market quarterly by 5% from 1/1/93 to 30/6/95.

The inputs of the ANN stock selection system are the seven inputs chosen in the earlier section and the output will be the return differences between the stock and the market return (excess returns). This is to enable the ANN to establish the relationships between inputs and the output (excess returns).

The training data set includes all data available until the quarter before the testing quarter. This is to ensure that the latest changes in the relationship of the inputs and the output are being captured in the training process.

In order to ensure the generalization ability of the ANN in selecting top performing stocks as well as its ability to perform consistently over time, sufficient training is important. The data used for the selection process are from 3rd quarter of 1995 (1/7/95-30/9/95), the 4th quarter of 1995 (1/10/95-31/12/95), 1st quarter of 1996 (1/1/1996-31/3/1996), 2nd quarter of 1996 (1/4/1996-30/6/1996), 3rd quarter of 1996 (1/7/1996-30/9/1996), 1/10/1996-31/12/1996 (1/10/1996-31/12/1996).

The testing inputs are being injected into the system and the predicted output will be calculated using the established weights. After which, the top 25 stocks with the highest output value will be selected to form a portfolio of stocks. These 25 stocks are the top 25 stocks recommended for purchase at the beginning of the quarter. Generalization ability of the ANN will be determined by the performance of the portfolio, measured by excess returns over the market as well as the percentage of top performers in the portfolio as compared to the benchmark portfolio (testing portfolio) at the end of the month.

Design 2 (Moving-Window System)

The Basic System is constrained by meeting the minimum sample size required for training process. However, this second design is going to forgo the recommended minimum sample size and introduce a moving-window concept. This is to analyze the ANN ability to perform under a restricted sample size environment.

The input and output variables are identical to that of the Basic System but the training and testing samples are different. The Moving-Window System uses three quarters as training samples and the subsequent quarter as the testing sample. The selection criteria are also identical to that of the Basic System in research Design 1.

Results

The ANN is made to train with 10,000 and 15,000 cycles. The reason for using these numbers of cycles for training is because the error converging is generally slow after 10,000, thus suggesting adequate training. Moreover, it does not converge beyond 15,000. This is an indicator that the network is overtrained (see Figure 1).

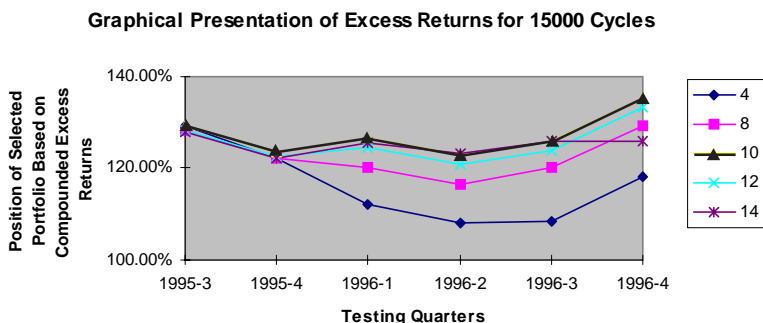
On a Pentium 100 Mhz PC, the training of four hidden neurons for 10,000 cycles takes approximately 1.5 hours, eight hidden neurons takes about 3 hours, and the most complex (14 hidden neurons) took about 6 hours. As for those architectures that require 15,000 cycles, it usually takes about 1.5 times the time it takes to train the network for 10,000 cycles.

The results of the Basic System based on the training and testing schedules mentioned are presented in two forms: (1) the excess return format and (2) the percentage of the top performers in the selected portfolio. These two techniques will be used to assess the performance and generalization ability of the ANN.

Testing results show that the ANN is able to “beat” the market overtime, as shown by positive compounded excess returns achieved consistently throughout all architectures and training cycles. This implies that the ANN can generalize relationships over time. Even at the individual quarters’ level, the relationships between the inputs and the output established by the training process is proven successful by “beating” the market in six out of eight possible quarters, which is a reasonable 75% (see Figure 2).

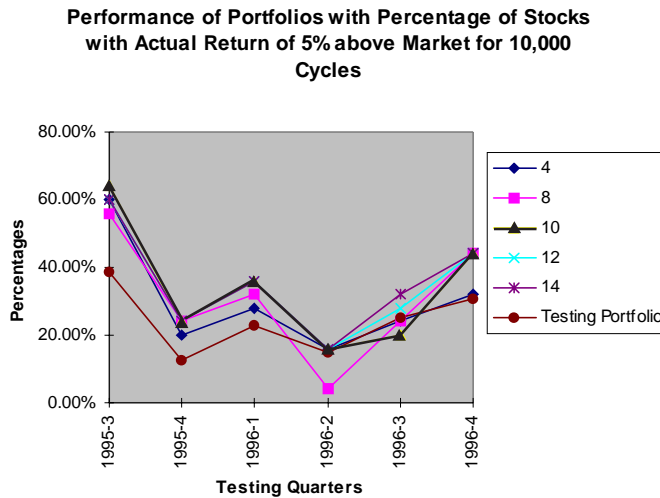
The Basic System has consistently performed better than the testing portfolio over time. This is evident by the fact that the selected portfolios have higher percentages of top

Figure 1. ANN Training at 15,000 cycles



Note: Numbers in key represent the number of hidden layer neurons

Figure 2. Performance of ANN above market



Note: Numbers in key represent the number of hidden layer neurons

performing stocks (above 5%) than the testing portfolio over time (see Figure 3). This ability has also enabled the network to better the performance of the market (SESALL Index) presented earlier.

The Moving-Window System is designed to test the generalization power of the ANN in an environment with limited data.

The generalization ability of the ANN is again evident in the Moving-Window System, as it outperformed the testing portfolio in 9 out of 13 testing quarters (69.23%). This can be seen in the graphical presentation where the line representing the selected portfolio is above the line representing the testing portfolio most of the time (see Figure 4). Moreover, the compounded excess returns and the annualized compounded excess returns are better than that of the testing portfolio by two times over. The selected portfolios have outperformed the market 10 out of 13 (76.92%) testing quarters and excess returns (127.48% for the 13 quarters and 36.5% for the Annualized compounded return) which proved its consistent performance over the market (SESALL index) overtime.

The selected portfolios have outperformed the testing portfolio in nine quarters (69.23%) and equal the performance in 1 quarter. This further proves the generalization ability of the ANN. Moreover, the ability to avoid selecting undesirable stocks is also evident by the fact that the selected portfolios have less of this kind of stocks than the testing portfolio in 10 out of 13 occasions (76.92%).

From the experimental results, the selected portfolios outperformed the testing and market portfolios in terms of compounded actual returns over time. The reason is because the selected portfolios outperform the two categories of portfolios in most of the testing quarters, thus achieving better overall position at the end of the testing period (see Figure 5).

Figure 3. Excess return of ANN-selected portfolio

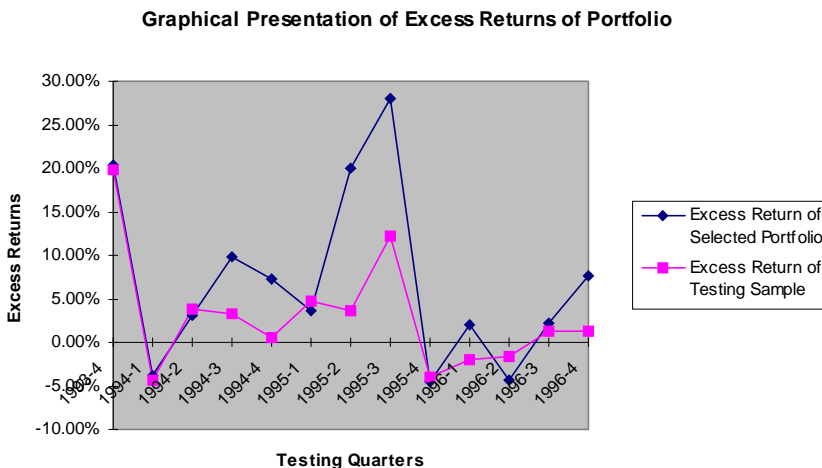
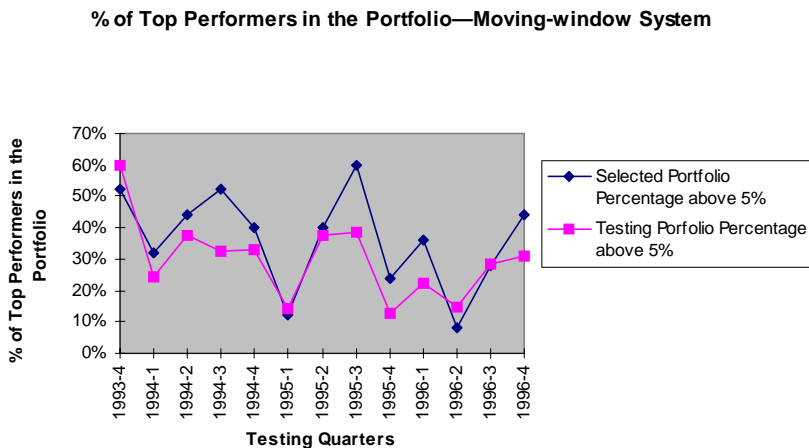


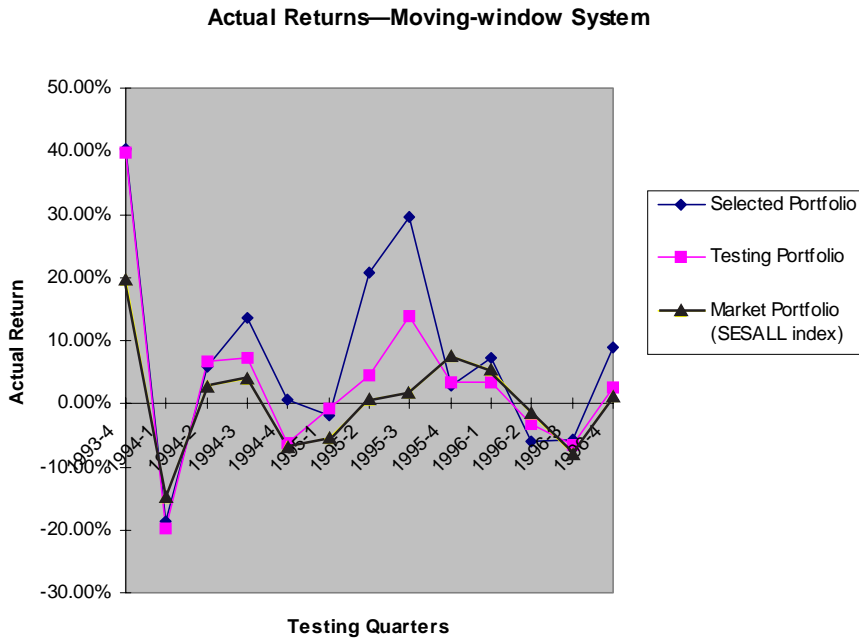
Figure 4. Excess return of ANN-selected portfolio (Moving-Window System)



Conclusion and Future Works

The ANN has demonstrated its generalization ability in this particular application. The results clearly show that improved returns are achieved when an ANN is used to pick out the better stocks. This is evident through the ability to single out performing stock counters and having excess returns in the Basic System over time. Moreover, the neural network has also showed its ability in deriving relationships in a constrained environment in the Moving-Window System, thus making it even more attractive for applications

Figure 5. Comparing performances



in the field of Finance. This work is a preamble for a stock recommendation system that can assist fund managers in getting better returns for portfolios managed by them. For individual investors, an ANN system will be able to provide a guide for wiser selection of counters, thus achieving better wealth growth.

For this work, the modeling horizons chosen were mainly short-term. While the results are true for short-term forecasting, they might not hold for longer-term price movements. Investors with a long-term view to investing might not find our neural network helpful. Therefore, an area of future research would be to investigate the longer-term modeling of stock counters.

As a related point, as this work focuses on short-term modeling, we have opted not to use any macroeconomic variables as they are collected, at best, only on a monthly basis and thus might be inappropriate for daily predictions. However, ignoring macroeconomic variables also has its own perils as these variables definitely have an impact on the stock markets. Perhaps to compensate for the slower release of these variables, future researches can instead use leading economic indicators to forecast the stock prices.

Also, this work is largely constrained by the availability of data. Therefore, when more data is available, performance of the neural networks can be better assessed in the various kinds of market conditions — such as bull, bear, high inflation, low inflation, or even political conditions — each of which has a different impact on stocks.

Last but not least, as researchers are discovering more powerful neural architectures at a fast pace, it is good to repeat the experiments using several architectures and compare the results. The best performance structure may then be employed.

References

- Bloomberg. (2005). Bloomberg Financial Database. Retrieved from <http://www.bloomberg.com/>
- Cao, Q., Leggio, K. B., Schniederjans, M. J. (2005). A comparison between Fama and French's model and ANNs in predicting the Chinese stock market. *Computers and Operations Research*, 32(10), 2499-2512.
- Chen, A. S., Leung, M. T., & Daouk, H. (2003). Application of neural networks to an emerging financial market: Forecasting and trading the Taiwan stock index. *Computers and Operations Research*, 30(6), 901-923.
- Credit Lyonnais Securities. (2005). *Credit Lyonnais Securities Financial Database*. Retrieved from <https://www.lcl.fr/>
- Gately, E. (1996). *Neural networks for financial forecasting—Top techniques for designing and applying the latest trading systems*. New York: John Wiley & Sons.
- Grudnitski, G., & Osburn, L. (1993). Forecasting S&P and gold futures prices: An application of neural networks. *The Journal of Future Market*, 13(6), 631-643.
- Halquist, C., & Schmoll, G. (1989). Neural networks: A trading perspective. *Technical Analysis of Stocks and Commodities*, 7(11), 48-54.
- Min, Q. (1999). Nonlinear predictability of stock returns using financial and economic variables. *Journal of Business & Economic Statistics*, 17(4), 419-429.
- Min, Q., & Maddala, G. S. (1999). Economic factors and the stock market: A new perspective. *Journal of Forecasting*, 18, 151-166.
- Olson, D., & Mossman, C. (2003). Neural network forecasts of Canadian stock returns using accounting ratios. *International Journal of Forecasting*, 19(3), 453-465.
- Perez, J. V., Torra, S., & Andrada, F. J. (2005). STAR and ANN models: Forecasting performance on the Spanish IBEX-35 stock index. *Journal of Empirical Finance*, 12(3), 490-509.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning the internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing, Vol. 1 and 2*. Cambridge, MA: MIT Press.
- Singapore Exchange. (2006). Singapore Exchange Limited (SGX). Retrieved from <http://www.ses.com.sg/>
- Trippi, R. R., & DeSieno, D. (1992). Trading equity index futures with a neural network. *The Journal of Portfolio Management*, 19(1), 27-33.
- Tsaih, R., Hsu, Y., & Lai, C. (1998). Forecasting S&P 500 index futures with a hybrid AI system. *Decision Support Systems*, 23, 161-174.

SECTION III:
ANNs in Manufacturing

Chapter X

Neural Networks in Manufacturing Operations

Eldon Gunn, Dalhousie University, Canada

Corinne MacDonald, Dalhousie University, Canada

Abstract

This chapter provides some examples from the literature of how feed-forward neural networks are used in three different contexts in manufacturing operations. Operational design problems involve the determination of design parameters, such as number of kanbans, in order to optimize the performance of the system. Operational-system decision support refers to the use of neural networks as decision-support mechanisms in predicting system performance in response to certain settings of system parameters and current environmental factors. Operational-system-control problems are distinguished from decision support in that the consequences of a control decision are both an immediate return and putting the system in a new state from which another control decision needs to be taken. In operational control, new ideas are emerging using neural networks in approximate dynamic programming. Manufacturing systems can be very complex. There are many factors that may influence the performance of these systems; yet in many cases, the true relationship between these factors and the system outcomes is not fully understood. Neural networks have been given a great deal of

attention in recent years with their ability to learn complex mappings even when presented with a partial, and even noisy, set of data. This has resulted in their being considered as a means to study and perhaps even optimize the performance of manufacturing operations.

This chapter provides some examples from the literature of how neural networks are used in three different contexts in manufacturing systems. The categories (1) operational design, (2) operational decision-support systems, and (3) operational control are distinguished by the time context within which the models are used. Some examples make use of simulation models to produce training data, while some use actual production data. In some applications, the network is used to simply predict performance or outcomes, while in others the neural network is used in the determination of optimal parameters or to recommend good settings. Readers who wish to explore further examples of neural networks in manufacturing can examine Udo (1992), Zhang and Huang (1995), and Wang, Tang, and Roze (2001).

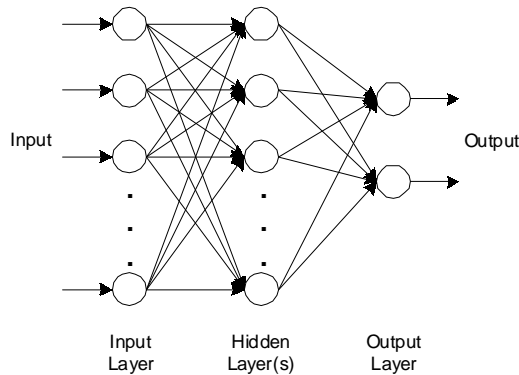
We begin with two areas in which neural networks have found extensive use in manufacturing. Operational-system design has seen considerable use of neural networks as metamodels that can stand in place of the system, as we attempt to understand its behavior and optimize design parameters. Operational-system decision support refers to the use of neural networks as decision-support mechanisms in predicting system performance in response to certain settings of system parameters. We close with a short introduction to an area where we anticipate seeing growing numbers of applications, namely the use of approximate dynamic programming methods to develop real-time controllers for manufacturing systems.

Operational-System Design Using Neural Networks

In the design of manufacturing operations, there are usually several performance measurements of interest, such as throughput, average work-in-process inventory WIP, or machine utilization. These measures are interrelated and sometimes conflicting. There may also be several design variables, such as number of kanbans or buffer sizes at each station, which will influence these measurements. Because of the complexity of these systems, simulation models are used to estimate system performance given a set of design values. Depending on the number of input variables, and the number of values that those variables could take on, the number of variable combinations can be so large that simulating all of them is not practical or even possible. Therefore, further work is necessary to ascertain the set of design parameters that will lead to the desired or optimal system performance.

Simulation optimization techniques (Andradottir, 1998; Fu, 2002; Tekin & Sabuncuoglu, 2004) have been employed in the search for the best set of design parameters. However, what may be a preferable approach is to develop a simulation metamodel. Metamodels are constructed to approximate the functional relationship between the inputs and

Figure 1. Feed-forward neural network



outputs of the simulation model. The particular approach we want to look at is the use of feed-forward neural networks (Figure 1) as simulation metamodels. Barton (1998) gives a broader overview of simulation metamodels, including response surface modeling. The main idea is that networks are trained on a subset of possible design parameter combinations using the resulting performance measurements obtained from the simulation model.

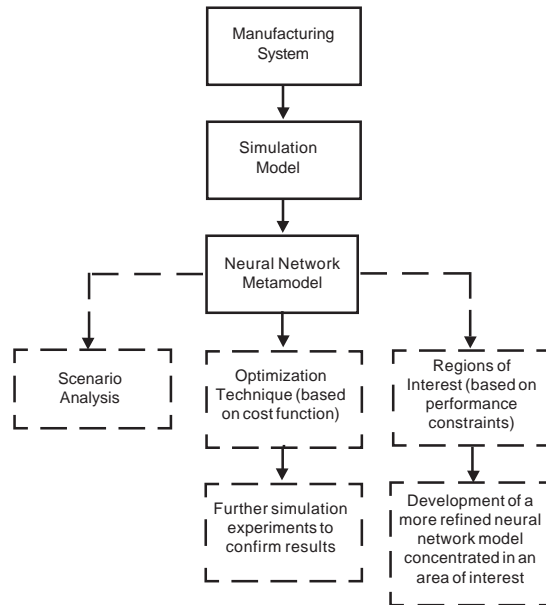
In the next two sections, different uses of neural network metamodels are described. One involves approximating the relationship between design parameters and system performance, and the other involves using a neural network metamodel of the inverse of the simulation model. Our focus in the discussion is on the use of the neural network as a metamodel. We do not discuss the process of training, although several of the references we cite discuss how their particular network is trained.

Neural Networks as Simulation Metamodels

The starting point is a simulation of a manufacturing system operating under a certain set of design parameters. Then a neural network can be trained to estimate the performance measurements (outputs). Once trained, the networks may then be used to perform scenario analysis rather than using the original simulation model. The network model may also be used to determine an optimal set of input parameters, based on minimizing (maximizing) a single output or a cost function of multiple outputs. Further simulations may be carried out near the “optimal solution” to validate the result. The network metamodel may also be used to identify input regions of interest, where the outputs satisfy a set of constraints, and more in-depth analysis of these regions should be carried out.

Hurrion (1997) developed a method for finding the optimal number of kanbans in a manufacturing system using a neural network as an alternative to simulation-optimization techniques. The example system used in this case consisted of two manufacturing

Figure 2. Neural network metamodeling approaches



cells that produced three intermediate parts and four finished products. Demand for each product arrived according to a Poisson process, and processing times at both cells varied by component. The system was to be controlled using kanbans, and therefore the problem was to determine the number of kanbans to be assigned to the seven stock points. The system was to be evaluated using a composite cost function, which was a function of the average WIP and the product-delay time (defined as the amount of time between the receipt of an order and the completion of the product). A Visual Interactive Simulation (VIS) model of a manufacturing system was built and run under different configurations, and a feed-forward neural network was trained using backpropagation. Inputs to the network consisted of 7 nodes (one for each kanban), and 10 output nodes consisting of an upper and lower confidence interval for the five response variables. These were the product-delay times for the four products and the average WIP.

The network was then used to evaluate all possible combinations of kanban allocations and determine the combination that minimized the cost function. The solution given by the network demonstrated that the cost function was fairly flat in the region of this solution. This solution and all other feasible adjacent solutions were further evaluated using the original simulation model, and a statistical test was conducted to determine the best solution amongst this set. This technique was repeated for two further iterations until the final optimal solution was determined.

The author acknowledged that a Response Surface Methodology (RSM) could have been used, but would only have been valid for the cost function used. In this approach,

if a different cost function were to be evaluated, the neural network developed by this technique could be used, as only the last procedure need be repeated.

In an extension to the Hurrion (1997) study, Savsar and Choueiki (2000) developed a “Generalized Systematic Procedure (GSP)” for determining the optimal number of kanbans in a just-in-time (JIT) controlled production line. At each station of the production line, there are two different types of kanbans required for each product the station produces: production ordering kanbans, which authorize production at the preceding station, and withdrawal kanbans, which show the number of parts that the subsequent station must withdraw.

The GSP involves six steps: (1) determining all possible Kanban combinations and selecting a limited set of these combinations to represent the space; (2) developing a simulation model of the system (including identifying the performance measures of interest) and simulating under each of the combinations chosen in the first step; (3) developing an objective function involving performance measurements of the system such as WIP and delay time; (4) training a neural network model with the outcome of the objective function as the target output of the network; (5) validating the neural network model; and (6) using the neural network to evaluate all possible kanban combinations with respect to the objective function and to determine the optimal combination.

The case problem used to illustrate the procedure was an example from an electronics production line. The serial production line had five stations, with random (Erlang-distributed) processing times and demand arrivals. The possible combinations were limited to a total of 5 kanbans at each station and a total of 25 kanbans for the entire system. Even with this limitation, the number of possible kanban combinations was determined to be 100,000. Therefore, a combination of knowledge and factorial design was used to limit the training dataset to 243 kanban combinations. The simulation model was developed and run with all of these kanban combinations, and the resulting WIP and delay times were observed. A cost function, which was a weighted combination of delay and WIP costs, was determined, and the total cost of each of the 243 kanban combinations tested was calculated. A feed-forward neural network consisting of 10 input nodes (one for each type of kanban at each of the five stations), six hidden nodes, and one output node (system cost) was built and trained using backpropagation. A regression model was then constructed using the same training data as was used to train the network. The network was shown to outperform the regression model in its ability to interpolate accurately within the design space. Finally, the network was used to test all possible kanban combinations, and the combination with the lowest total cost was determined. Other possible combinations with total costs less than 1% different than the optimal solution were also determined, although no further evaluation of these possible solutions was performed.

For an example of a printed circuit board (PCB) manufacturing plant, Chen and Yang (2002) developed a neural network metamodel using simulation. Given a goal of maximizing the yield of the plant, they first developed a neural network metamodel of the system, with lot size, degree of line balance in the system, mean time between failures of machines, mean time to repair, time limit of the paste life, and capacity of the input buffer as the inputs to the model, and yield as the output. Once a BP network model had been trained, the resulting function was maximized — subject to constraints on the inputs — using a

simulated annealing optimization method. The best solution produced by their technique was better than that of Shang and Takikamalla, who had used a regression metamodel and RSM for the same problem (as cited in Chen and Yang [2002]).

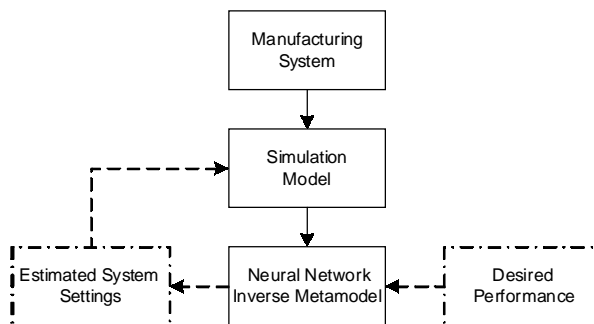
Some other examples include Altiparmak, Dengiz, and Bulgak (2002), who also used a neural network and simulated annealing approach to determine the buffer size configuration that would produce the highest production yield in an asynchronous assembly system. Chambers and Mount-Campbell (2002) use neural networks to predict the sojourn time of jobs at each station in a manufacturing system given the buffer sizes at each station and other information about the system. They then use the network to determine the optimal buffer sizes in order to minimize the sojourn time while maintaining the desired product mix. Markham, Mathieu, and Wray (2000) compared the use of neural networks and decision trees in kanban setting for a manufacturing shop. They found that neural networks were comparable to classification and regression trees in predicting the optimal number of kanbans.

Inverse Neural Network Metamodels

If a simulation can be seen as a function from a set of input parameters to an output, this raises the question of the existence of an inverse function that maps the system performance to the inputs that produce it. A number of authors have attempted to train a neural network metamodel to approximate the inverse function of the simulation model. This is done by using the performance measurements from the simulation model as the inputs in the training data and the corresponding system-parameter settings as the output. Validation of the results may be achieved by testing the solution provided by the neural network in the original simulation model. This approach has obvious challenges, since several different settings of the simulation inputs may have the same (approximately) simulation outputs so that the inverse is not in fact a function (i.e., 1-1).

Chryssolouris, Lee, Pierce, and Domroese (1990) used a neural network to determine the appropriate number of resources to assign to work centers within a job shop. A simulation model was used to produce the dataset used in training the neural network. The

Figure 3. Inverse neural network metamodel



simulation model produced performance measures for the system, such as mean tardiness and mean resource utilization, given a certain number of resources allocated at each station, work load, and operational policy. The authors then trained a multilayer perceptron neural network, using the performance measures as the input, and the number of resources at each station as the output. Thus, the role of the network was to act as the inverse to the simulation. Once trained, the desired performance measurements can be entered, and the network predicts the appropriate number of resources at each station to achieve the desired result. The authors noted the importance of using contrary type performance measures, and the desirability of screening criteria to eliminate poor design combinations, and only training the network with good results.

Using a similar methodology, Mollaghasemi, LeCroy, and Georgiopoulos (1998) developed a Decision Support System (DSS) for determining the optimal allocation of resources and queueing strategy in the test operation of a major semiconductor manufacturing company. The operation involved three different types of testers, and the goal was to determine the number of each type of tester and the type of queueing strategy (FIFO, SPT, highest demand, or lowest slack) in order to achieve desired performance measurements such as cycle time, WIP, and utilization of the three different types of testers. A simulation model of the operation was used to generate the estimates of performance for different combinations of resource allocations and queueing strategies.

Because the feed-forward neural network was to be used as the inverse to the simulation model, the network was trained using the performance outcomes as inputs and the corresponding design values (number of testers, type of queueing strategy) as the output. For each performance measurement, five ranges of possible values were determined, and five input nodes were used to represent these ranges — if the value of the measurement fell in a particular range, the corresponding input was set to one, otherwise it was zero. As well, the output nodes of the network represented each possible value for each design parameter; for example, since there were four possible queueing strategies, four output nodes were assigned. The training data was transformed such that if a particular strategy were used in the example, the node was set to one; otherwise, it was set to zero. The resulting network had 25 input nodes and 13 output nodes. The network was trained using backpropagation.

Once the network was trained, the desired combination of performance ranges (WIP, cycle time, etc.) could be presented as input to the network, and the network would then identify the corresponding combination of design parameter values to achieve that performance. The authors tested the network by presenting a combination of performance outcomes that were not included in the original training set. The outputs of the network were not exactly zero or one. Therefore, for each set of output nodes, the nodes with values closest to one were chosen as the solution. The authors acknowledged that if the network was presented with a combination of performance measurements that were infeasible (given the conflicting nature of these measures), the neural network would still produce a set of operational parameters; therefore, once such a result is obtained, it should be further tested with the original simulation model to ensure it is an achievable result.

The use of an inverse neural network metamodel does assume that an inverse of the performance function exists, which may not be the case. There may be multiple combi-

nations of design parameters that would result in the desired outcomes. This technique would only produce one such solution. Further investigation using the original simulation model, as previously mentioned, would normally be required.

Operational Decision Support Using Neural Networks

Another common situation in industrial operations is that the outcome of a manufacturing process is often influenced by many variables; however, only a subset (perhaps only one) of these variables is controllable. Operators must determine the “right” setting for a process or course of action to take, given information about the current situation, in order to achieve desired outcomes or to produce a “good” product. Sometimes the decision can be made based on previous experience, but often trial and error is necessary when previously unseen combinations of values are encountered. Neural networks can be used to assist in the decision making process by being trained to learn the relationship between these variables and process outcomes. The data used for this training may be taken from previously collected process data, or collected through experimentation. The trained network may then be used to test certain settings to determine the output, or may be further analyzed to provide the optimal setting or best decision given the current situation.

Coit, Jackson, and Smith (1998) demonstrate the use of neural networks in two industry examples; wave soldering and slip casting. In the wave soldering example, the problem was to develop a model to determine the best process settings (i.e., preheater temperature and belt speed) for the wave soldering machine in order to minimize the number of solder connection defects in the printed circuit boards (PCBs). Each PCB had several design characteristics, such as mass, size, and component density, and the firm produced many different models. As the thermal condition of the card when it enters the solder wave was considered the most important determinant of the soldering quality, this data was collected using special testing apparatus over a period of two months. Thermal condition of a card is described by the mean temperature, standard deviation, and temperature gradient at the wave.

Figure 4. Manufacturing process decision problem

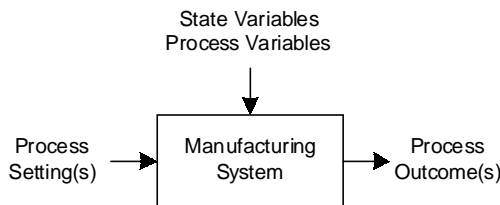
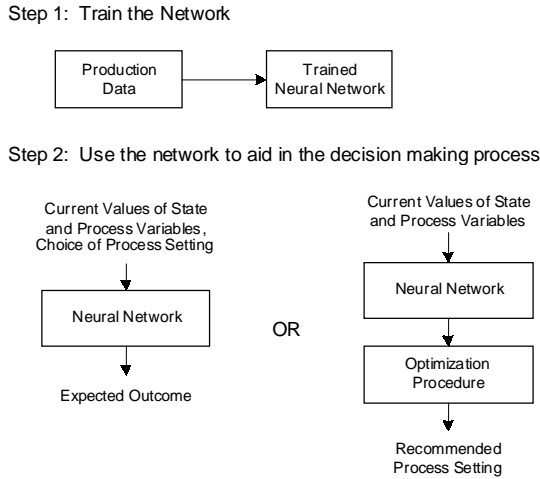


Figure 5. Neural network decision support



Three feed-forward neural networks, each with one output node, were constructed to predict the thermal condition (mean temperature, standard deviation and gradient) based on the design parameters of the PCBs and the process settings. Each network consisted of 14 inputs: design inputs, such as length of the card, card mass, and thickness; and process settings, which included four preheater temperatures and the belt speed. Finally, another neural network was constructed to use the thermal predictions as input and predict the category of solder quality (i.e., excellent, good, or fair).

Coit, Jackson, and Smith (1998) also detail the use of neural networks in predicting quality in a slip-casting process. While slip casting allows for the production of complex shapes such as sinks or statues, it is difficult to produce products that are free from defects, given the number of variables that can affect the quality of the casting. In order to avoid fractures and/or deformities in a casting, the moisture gradient within the casting should be as uniform as possible. As well, another output measurement of the process is casting rate, which is the thickness of the cast achieved during the casting time; the higher the casting rate, the less time the cast must spend in the mold.

In this application, neural networks were used to predict the moisture gradient and casting rate given ambient conditions (relative humidity and temperature), the casting time and properties of the slip (such as moisture content, viscosity, and temperature). The manufacturer had substantial production data with these parameters, and additional experiments were conducted to measure the effects of extreme values of the ambient temperature, humidity, and sulfate content in the slip. In all, ten slip variables, the two ambient or state variables, and the casting time were used as input to two feed-forward neural networks with either moisture gradient or casting rate as the single output. Lam,

Petri, and Smith (2000) discuss the process improvement module and the fuzzy-logic expert system which used these neural networks to recommend the best set of controllable variables and casting times. This system has been implemented at a major U.S. plant.

Another example is that of Philipoom, Wiegmann, and Rees (1997) in the assignment of due dates to jobs arriving at a shop, where there is work in process, and the processing times at each stage of production are random variables. The goal is to assign a due date (date for completion as quoted to the customer) to an arriving job that will minimize the expected penalty cost due to early or late completion. Here, the authors assume that the cost for late completion is different than for early completion. Three different shop configurations were simulated, and then 23 job-specific and shop-specific characteristics were measured each time a job entered the simulated shop. The departure time of each job was also measured. A neural network was then trained using the 23 characteristics as input and the departure times as outputs. Mathematical programming and OLS regression techniques were also used to predict the completion time of each incoming job. Each model was presented with new data, and then the difference between the predicted completion date (assigned due date) and the simulated completion date were used to calculate the penalty cost incurred for each job. Overall, the neural network performed as well or better than the other techniques tested for these examples.

Schlang et al. (1997) reported on the use of neural networks in the steel industry. One application was in a wide-strip hot-rolling process, where steel sheets underwent prerolling in a roughing mill prior to being brought to a final thickness by a finishing mill. The width of the sheets could only be controlled at the roughing stage, although material characteristics and the state of the finishing mill were also known to be factors in the final width of the sheets after the finishing stage. Due to the variability in processing, a safety margin of several millimeters was used to ensure the sheet was not too narrow after finishing. Any excess width after the finishing stage was trimmed off and recycled. A neural network was built and trained to predict the width of a sheet of steel after the finishing stage, given such information as the material composition, material temperature, and finishing mill parameters, and also the initial settings at the prerolling stage. Because the network could more accurately predict the final width of the sheet, the safety margin could be reduced, therefore reducing the amount of recycling.

Kilmer, Smith, and Shuman (1999) developed parallel neural networks as metamodels for discrete event simulations. They modeled an (s,S) inventory system and determined the expected system cost and variance, given selected values of setup cost, stockout cost, and values of s and S. Two neural networks were then trained; one with the expected system cost as the output and the other with the variance of the average cost from multiple replications of the simulation at each point. These estimates were then used as confidence intervals for the expected total cost, and shown to closely replicate results from the simulation model itself when tested on data points not originally in the training set.

Sabuncuoglu and Touhami (2002) estimate manufacturing system performance using neural networks. They experimented with both simple and complex systems, and with using deterministic and stochastic processing times and interarrival times. In these experiments, a due date for an arriving job is determined based on the total work content of the job multiplied by a tightness factor. The simulation models were run with varying interarrival times, tightness factors, and queue waiting discipline (shortest processing time, earliest due date, or modified operation due date). The mean machine utilization,

mean job tardiness and mean job flow time for each job were recorded. A feed-forward neural network was then trained (using backpropagation with momentum) for each experiment with the interarrival time, tightness factor and queue-waiting disciplines as inputs, and one of the performance measurements as the output.

Huang, et al. (1999) examined the use of neural networks to predict the WIP levels and throughput for operation stages of a wafer-fabrication process. The goal was to develop a network which could predict the WIP level and throughput of an operation stage in the next time period, given information on the current situation, so that managers could proactively implement corrective actions. They determined through testing that using the current WIP levels and throughput at an operation stage, as well as the same levels from the two upstream operation stages, as inputs to a feed-forward neural network trained with backpropagation provided the best prediction results. They recommended a two-stage procedure for the implementation of the network. The predicted WIP levels and throughput from the network were compared to the standard performance measures and represented as “high,” “normal,” or “low.” Predictions other than “normal” indicated that managers needed to implement corrective actions.

The ability of feed-forward neural networks to approximate the functional relationship between input and output variables, even with incomplete data, is very useful in this application. One perhaps obvious caution is that a trained network is only valid for the process data on which it was trained. The examples presented earlier are static, in that it is assumed that the system itself does not change. If the process is changed, the network should be retrained. Since processes may change over time, periodic testing and/or retraining should also be implemented.

Operational System Control Using Neural Networks

Control problems are distinguished from decision support in that the consequences of a control decision are both an immediate cost or benefit and the fact that the system is now in a new state from which another control decision needs to be taken. The view of manufacturing systems as a manufacturing-control problem has been current for some time now (Gershwin, Hildebrandt, Suri, & Mitter, 1986). Typically, these control systems are challenging to optimize.

Although the focus of this chapter is on feed-forward networks used as approximators, it is worth noting that recurrent neural networks have been developed (Rovithakis, Gaganis, Perrakis, & Christodoulou, 1999) to provide controllers for quite complex manufacturing systems. The aim of the controller is to maintain the system at prescribed buffer levels (WIP levels) in a stable manner. In Rovithakis, Perrakis, and Christodoulou (2001), an application of this neural network control is reported to perform well on a real manufacturing system aimed at controlling WIP levels for 18 product types in a job-shop environment.

A more general approach to manufacturing control will not seek to maintain fixed WIP levels but instead seek to optimize an objective function of some sort. A natural approach to the manufacturing control problem has been dynamic programming, but typically most real manufacturing systems have such large state spaces that render this approach infeasible. However, ideas developed in neurodynamic programming (Bertsekas & Tsitsiklis, 1996), also known as reinforcement learning (Sutton & Barto, 1998), have begun to have important applications in manufacturing and we anticipate considerable growth in these types of applications. It will thus be useful to sketch out how these ideas work in order to see the important role that feed-forward neural networks play. Bertsekas (2001) gives a nice summary article on neurodynamic programming that is the basis for much of the discussion that follows.

The basic finite horizon dynamic programming model can be summarized as follows. We have a dynamic system that evolves according to $x_{t+1} = f(x_t, u_t, w_t)$, where x_t is the system state, u_t is a control action, and w_t is a random disturbance. For an infinite horizon control problem with discrete time periods, the task facing us is to compute the state cost-to-go

function $J(x)$, which solves Bellman's Equation. The goal is to maximize $E \left[\sum_{t=1, T} g_t(x_t, u_t, w_t) \right]$

by taking allowable control actions $u_t \in A(x_t)$. The $g_t(x_t, u_t, w_t)$ is a reward earned on the t^{th} stage of this process.

The dynamic programming approach consists of attempting to solve the recursive equation $J_t(x_t) = \text{Max}_{u \in A(x_t)} \{ E [g_t(x_t, u_t, w_t) + J_{t+1}(x_{t+1})] \}$. Taking this to a control model, with stationary reward functions $g()$, and stationary probability functions, the Bellman's Equation is:

$$J(x) = \text{Max}_{u \in A(x)} \{ E [g(x, u, w) + \alpha(x, u, w) J(x')] ; x' = f(x, u, w) \} \quad (1)$$

The problem as portrayed here is the minimization of discounted rewards with a discount factor $\alpha(x, u, w)$. The dependency on the x, u , and w is meant to indicate that this procedure can deal with both standard markov processes but also semi-markov processes where the transition time depends on the state, the decision, and the random variable. The use of discounting to deal with the infinite horizon is simple for exposition. However, an objective that maximizes average rewards per unit time is equally possible. Both Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998) provide the details that are lacking above. Das, Gosavi, Mahadevan, and Marchellack (1999) develop a generic algorithm for the use of reinforcement learning techniques to minimize average costs in a semi-markov decision setting.

There are two basic approaches to solving the DP model: value iteration and policy iteration. There are several types of approximation that form the basis of NDP/RL approaches. The simplest is approximation to the cost-to-go function. Bellman equations can be thought of as a recursive approach to learning the value of the cost-to go function $J()$. Algorithms based on this are known as value iteration. The problem is that the number of possible state variables is very large (often continuous). This means it is impossible

to evaluate the $J(x)$ for all possible x . A natural approach is to use a neural network $J(x, r)$ where r are the weights of a feed-forward neural network to fit the current estimates $J'(x_i)$, $i = 1, N$. This neural network can be used to update the estimates J' again using:

$$J'(x) = \underset{u \in A(x)}{\text{Max}} \{E[g(x, u, w) + \alpha(x, u, w)J(x', r)]\}; x' = f(x, u, w) \quad (2)$$

and a new neural network fit to the current J' . The estimates of J' can be updated either synchronously (all of the states x) or asynchronously (selected states).

A second approach is that of Q -learning. The Q factor is the function $Q(x, u)$ given by:

$$Q(x, u) = E[g(x, u, w) + \alpha(x, u, w)J(x')]; x' = f(x, u, w) \quad (3)$$

There are several approaches to estimating the Q -factors. One is to use a neural network to approximate J and then to estimate the expected value by simulation. Another is to estimate J itself by simulation of a heuristic policy and use this simulation to estimate $Q(x, u)$. There are a broad variety of ways to build up approximations of $Q(x, u)$ using a variety of learning algorithms. However, in most practical cases, the dimension of the state and action space is again so large that the Q -factors can be evaluated at only a small number of potential states. A feed-forward neural network can then be used to approximate Q as $Q(x, u; r)$ where r are again the weights of some appropriately chosen neural network architecture. (Note we are using r throughout this section to denote weights/parameters of the fitting architecture and are obviously not the same quantities from one type of approximation to the next. The meaning should be clear from the context). This gives $J(x) = \underset{u \in A(x)}{\text{Max}} \{Q(x, u; r')\}$ and the optimal control policy as $u(x) = \underset{u \in A(x)}{\text{arg max}} \{Q(x, u; r')\}$.

The third opportunity involves the approximation of the policy function itself. If we have a good estimate of the cost-to-go function $J()$ or a good estimate of the Q -factors, then an approximate policy is:

$$u(x) = \underset{u \in A(x)}{\text{arg max}} \{E[g(x, u, w) + \alpha(x, u, w)J(x', r)]\}; x' = f(x, u, w)$$

or

$$u(x) = \underset{u \in A(x)}{\text{arg max}} \{Q(x, u; s)\}.$$

Again it is often the case that there are such a large number of states that it is unrealistic to evaluate $u(x)$ at every state x . A possible solution is the development of a policy network $u(x; r)$ that approximates those states where the evaluation has been carried out.

In the quite common situation where only a reasonably small finite number of policy actions are available, this network becomes a classification network.

The SMART algorithm developed in Das, Gosavi, Mahadevan, and Marchellack (1999) applies reinforcement-learning theory in a case of semi-markov decisions and with an objective of average costs, not discounted. The technique is effectively building up an approximation of the Q factors through simulation and learning updates. Then, a feed-forward neural network is used to extend the estimates at the simulated states and actions to the entire state and action space. In Das, et al. (1999), the SMART algorithm is applied to large preventive maintenance problem (approximately 10^7 states). Paternina-Arboleda and Das (2001) give an application of SMART to develop control policies for serial production lines that are an improvement on conventional kanban, CONWIP, and other control policies.

Shervais, Shannon, and Lendaris (2003) use a neurodynamic-programming approach to a problem of physical distribution in the supply chain. They modeled the supply chain process with a simulation and then fit a neural network to the simulation results. From that point on, the neural network acted as the model of the system dynamics. Their approach is essentially a policy iteration algorithm. Any policy is developed at a finite set of state points and then extended to the entire space using a policy network.

Gershwin (1989) has pointed out the generality of the dynamic programming framework for the optimal control of manufacturing systems. This suggests that as the ideas of neurodynamic programming become better known in the manufacturing community, we will see increased use of this approach and a corresponding increase in the use of neural networks to approximate the cost-to-go function, the Q factors, and the policy functions.

Conclusion

The preceding sections provided some examples of how neural networks are being applied in manufacturing operations. Neural networks are well suited to manufacturing applications, which tend to be very complex with interrelated outputs. While the use of neural networks continues to grow, there are still some outstanding research issues in this area, as well as practical issues in their application.

The attraction of using neural networks has to do with the ability to map many inputs onto multiple outputs without knowing the underlying function. One practical issue in manufacturing is determining which inputs do in fact affect the outputs. For example, in decision-support systems, which of the environmental factors (e.g., plant temperature, type of tooling used, etc.) have a significant impact on the performance measurement of interest? Including input variables, which, in reality, do not affect the process outcomes, will over-complicate the network design and perhaps lead to difficulties during training. While expert opinion may be solicited prior to the construction of the network, sometimes the applicators will be looking to the network to determine whether or not there is any relationship. Unfortunately, trained feed-forward neural networks are like a "black box" and do not provide insight or general rules for the relationship between the input and the output of the network.

When there are a very large number of combinations of possible inputs, determining the correct composition of the training dataset is a challenge in itself. In order to ensure good generalization within the sample space, a broad range of sample points should be chosen. In the case of decision-support systems for manufacturing processes, where actual production data is used for training, it may be necessary to conduct experiments to capture observations for certain rarely seen or extreme combinations of process variables, as they may not exist in the historical production data. Otherwise, in future, one may be calling upon the network to extrapolate, which may lead to highly inaccurate results.

The architecture design of the networks remains a challenge. The principle of Occam's Razor would argue that simpler functions (fewer hidden nodes/layers) are more desirable, yet the resulting network may not fit the underlying function closely enough. Overfitting must also be addressed to ensure the network has good generalization properties. This is usually addressed by splitting the data into a training set and a testing set and stopping the training of the network when the error between the network output and the testing set begins to rise. While there are rules of thumb for determining the number of hidden nodes, often the approach is to test various architectures and choose the one that provides the best compromise between generalization and error minimization.

The use of metamodels does not guarantee an easy means to optimization. Even with an analytical function, optimization is not an easy problem and heuristic methods may be necessary to generate a good, but not necessarily optimal, solution.

Neural network approaches can bring designers and operators closer to the right design or decision faster than traditional approaches. Although careful design and testing can improve the accuracy of the network's approximations, the output is still in fact an approximation. Further testing and validation may be necessary before system implementation.

References

- Altıparmak, F., Dengiz, B., & Bulgak., A. A. (2002). Optimization of buffer sizes in assembly systems using intelligent techniques. In E. Yucesan, C. H. Chen, J. L. Snowdon, & J. M. Charnes (Eds.), *Proceedings of the 2002 Winter Simulation Conference, Vol. 2* (pp. 1157-1162).
- Andradottir, S. (1998). A review of simulation optimization techniques. In D. J. Medeiros, E. F. Watson, J. S. Carson, & M. S. Manivannan, (Eds.), *Proceedings of the 1998 Winter Simulation Conference* (pp. 151-158).
- Barton, R. R. (1998). Simulation metamodels. In D. J. Medeiros, E. F. Watson, J. S. Carson, & M.S. Manivannan (Eds.), *Proceedings of the 1998 Winter Simulation Conference* (pp. 167-174).
- Bertsekas, D. P. (2001). Neuro-dynamic programming: An overview. In C. A. Floudas, & P. M. Pardalos (Eds.), *Encyclopedia of Optimization*, 17 (pp. 17-22).
- Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena.

- Chambers, M., & Mount-Campbell, C. A. (2002). Process optimization via neural network metamodeling. *International Journal of Production Economics*, 79, 93-100.
- Chen, M. C., & Yang, T. (2002). Design of manufacturing systems by a hybrid approach with neural network metamodeling and stochastic local search. *International Journal of Production Research*, 40, 71-92.
- Chrysolouris, G., Lee, M., Pierce, J., & Domroese, M. (1990). Use of neural networks for the design of manufacturing systems. *Manufacturing Review*, 3, 187-194.
- Coit, D. W., Jackson, B. T., & Smith, A. E. (1998). Static neural network process models: Considerations and case studies. *International Journal of Production Research*, 36, 2953-2967.
- Das, T. K., Gosavi, A., Mahadevan, S., & Marchellack, N. (1999). Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45, 560-574.
- Fu, M. (2002). Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing*, 14, 192-215.
- Gershwin, S. B. (1989). Hierarchical flow control: A framework for scheduling and planning discrete events in manufacturing systems. *Proceedings of the IEEE*, 77(1), 195-209.
- Gershwin, S. B., Hildebrandt, R. R., Suri, R., & Mitter, S. K. (1986). A control perspective on recent trends in manufacturing systems. *IEEE Control Systems Magazine*, 6(2), 3-15.
- Huang, C. L., Huang, Y. H., Chang, T. Y., Chang, S. H., Chung, C. H., Huang, D. T., et al. (1999). The construction of production performance prediction system for semiconductor manufacturing with artificial neural networks. *International Journal of Production Research*, 37, 1387-1402.
- Hurriion, R. D. (1997). An example of simulation optimisation using a neural network metamodel: Finding the optimum number of kanbans in a manufacturing system. *Journal of the Operational Research Society*, 48, 1105-1112.
- Kilmer, R. A., Smith, A. E., & Shuman, L. J. (1999). Computing confidence intervals for stochastic simulation using neural network metamodels. *Computers & Industrial Engineering*, 36, 391-407.
- Lam, S. S. Y., Petri, K. L., & Smith, A. E. (2000). Prediction and optimization of a ceramic casting process using a hierarchical hybrid system of neural networks and fuzzy logic. *IIE Transactions*, 32, 83-91.
- Markham, I. S., Mathieu, R. G., & Wray, B. A. (2000). Kanban setting through artificial intelligence: A comparative study of artificial neural networks and decision trees. *Integrated Manufacturing Systems*, 11(4), 239-246.
- Mollaghasemi, M., LeCroy, K., & Georgiopoulos, M. (1998). Application of neural networks and simulation modeling in manufacturing design. *Interfaces*, 25(5), 100-114.
- Paternina-Arboleda, C. D., & Das, T. K. (2001). Intelligent dynamic control of single-product serial production lines. *IIE Transactions*, 33, 65-77.

- Philpoom, P. R., Wiegmann, L., & Rees, L. P. (1997). Cost-based due-date assignment with the use of classical and neural-network approaches. *Naval Research Logistics*, 44, 21-46.
- Rovithakis, G. A., Gaganis, V. I., Perrakis, S. E., & Christodoulou, M. A. (1999). Real time control of manufacturing cells using dynamic neural networks. *Automatica*, 35, 139-149.
- Rovithakis, G. A., Perrakis, S. E., & Christodoulou, M. A. (2001). Application of a neural-network scheduler on a real manufacturing system. *IEEE Transactions on Control Systems Technology*, 9, 261-270.
- Sabuncuoglu, I., & Touhami, S. (2002). Simulation metamodelling with neural networks: An experimental investigation. *International Journal of Production Research*, 40, 2483-2505.
- Savsar, M., & Choueiki, M. H. (2000). A neural network procedure for kanban allocation in JIT production control systems. *International Journal of Production Research*, 38, 3247-3265.
- Schlang, M., Broese, E., Feldkeller, B., Gramckow, O., Jansen, M., Poppe, T., et al. (1997). Neural networks for process control in steel manufacturing. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 1* (pp. 155-158).
- Shervais, S., Shannon, T. T., & Lendaris, G. G. (2003). Intelligent supply chain management using adaptive critic learning. *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems And Humans*, 33, 235-244.
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
- Tekin, E., & Sabuncuoglu, I. (2004). Simulation optimization: A comprehensive review on theory and applications. *IIE Transactions*, 36, 1067-1081.
- Udo, G. (1992). Neural networks applications in manufacturing processes. *Computers & Industrial Engineering*, 23, 97-100.
- Wang, J., Tang, W. S., & Roze, C. (2001). Neural network applications in intelligent manufacturing: An updated survey. In A. Kusiak, & J. Wang (Eds.), *Computational intelligence in manufacturing* (chap. 2). Boca Raton, FL: CRC Press.
- Zhang, H. C., & Huang, S. H. (1995). Applications of neural networks in manufacturing: A state of the art survey. *International Journal of Production Research*, 33, 705-728.

Chapter XI

High-Pressure Die-Casting Process Modeling Using Neural Networks

M. Imad Khan, Deakin University, Australia

Saeid Nahavandi, Deakin University, Australia

Yakov Frayman, Deakin University, Australia

Abstract

This chapter presents the application of a neural network to the industrial process modeling of high-pressure die casting (HPDC). The large number of inter- and intradependent process parameters makes it difficult to obtain an accurate physical model of the HPDC process that is paramount to understanding the effects of process parameters on casting defects such as porosity. The first stage of the work was to obtain an accurate model of the die-casting process using a feed-forward multilayer perceptron (MLP) from the process condition monitoring data. The second stage of the work was to find out the effect of different process parameters on the level of porosity in castings by performing sensitivity analysis. The results obtained are in agreement with the current knowledge of the effects of different process parameters on porosity defects, demonstrating the ability of the MLP to model the die-casting process accurately.

Introduction

HPDC is a process used to produce various structural elements for the automotive industry, such as transmission cases, engine sump, rocker covers, and so on. The process begins with pouring melted aluminum in the shot sleeve cylinder through a ladle. After the die is closed, the metal is pushed inside the die cavity by moving a plunger. The plunger starts initially with a low velocity, then the velocity increases during the piston's motion, and the velocity is decreased at the end when nearly all the liquid metal is injected into the die. The metal is injected through gate and runner system at a high velocity and pressure. The die is then opened and a robotic arm extracts the solidified part. The die is lubricated to facilitate the extraction of casting and to avoid soldering of the metal with the die surface. The extracted casting with a biscuit is then cooled down with water and is placed on a conveyer belt for further treatment or otherwise stored on a rack for quality-control tests.

The HPDC process is a complex process, consisting of over 150 inter- and intradependent process parameters. For example, there is a dependency between the gate velocity, the fill time, and the die temperature (Davis, 1978). If the fill time and the gate velocity are optimized, the die temperature becomes less critical. The interaction between the fill time and the metal pressure is also well-known (Walkington, 1990). The complexity of the process results in many problems like blistering and porosity. While the complexity of HPDC makes it difficult to obtain an accurate physical model of the process, having an accurate model of the die-casting process is paramount in order to understand the effects of process parameters on casting defects such as porosity.

Porosity is a defect in which the HPDC machine produces castings with pores in them as a result of either gas entrapment or vacuum due to poor metal flow at the location of pore occurrence. Porosity is by far the most highly occurring defect in automotive engine castings, resulting in the largest percentage of scrap of engine-component castings (Andresen & Guthrie, 1989). At the same time, porosity is one of the most difficult defects to eliminate in die casting. It is in the best interest of the industry (e.g., car manufacturers) and the consumer of die castings that porosity is eliminated completely from the castings, but this is not always possible to do with the current level of process understanding. The industry generally has to settle to move porosity to different noncritical locations in a casting rather than to remove it completely. In addition, attempts to eliminate porosity defects can affect other process settings and result in other casting defects.

Understanding of how HPDC process parameters influence casting defects such as porosity can eventually lead to determining the optimal process parameters to reduce the chance of defects occurring in the castings. The variety and often conflicting nature of the states of process parameters makes it hard in practice to achieve a globally optimized process with no defects in castings. Thus, the industry is generally opting for defect reduction on the basis of intended use of the casting; for example, a casting that has to be attached to other parts using bolts should not have weakness close to the bolt hole. It is crucial that there is either low or no porosity in the area close to the hole, while defects that lie in other parts of the same casting that does not affect structural integrity of the casting can be tolerated.

Background

The porosity defect can be divided into three major types, which are gas porosity, shrinkage porosity, and flow porosity. In HPDC, the first two types of porosity are mostly encountered. The gas porosity is the porosity in casting due to the presence of gas. This type can arise from gas produced during process, entrapped air, and melt composition. The shrinkage porosity is due to shrinkage of metal, so that the metal loses volume and hence more metal is required to fill the voids produced. In HPDC, it is hoped that this problem can be minimized with the application of high pressure to fill the voids when metal is in the solidification state. Formation of porosity in aluminium castings is a combination of die-casting process parameters, such as melt composition and solidification properties, under high pressure. Main process-related porosity formation mechanisms include solidification and entrapped-gas-related formation. Melt related porosity formation is of minor importance, primarily because hydrogen entrapment in HPDC is not a big problem (Walkington, 1997). Hydrogen entrapment can be a serious problem if there is a significant amount of scrap being remelted. The specific reasons for porosity formation are undesirable states of shot sleeve, cavity, runners, vent and gates, solidification pressure, lubricant quantity, and steam formation from water during the process.

Porosity formation is a subject of active research that can be divided into discussions of porosity types, whether gas or shrinkage (Garber & Draper, 1979) or the specific issues regarding porosity formation like entrapped air in shot sleeve or cavity (Garber, 1981, 1982; Thome & Brevick, 1995), gate and runner systems (Andresen & Guthrie, 1989), pressure (Kay, Wollenburg, Brevick, & Wronowicz, 1987; Murray, Chadwick & Ghomashchi, 1990), and melt composition (Kaghatil, 1987; LaVelle, 1962).

Shot-Related Porosity Formation

Shot-sleeve-related parameters are perhaps the most sensitive ones when it comes to entrapped-air porosity. The parameters like acceleration, stage velocities, diameter, or even deceleration are all shot-related parameters determining the formation of metal-wave patterns, which can be crucial factors in deciding whether air becomes entrapped. Other important parameters are shot-delay time and the percentage fill of the shot sleeve.

As soon as the metal is ladled, the goal of HPDC is to begin injection as soon as possible but still at the right time in the case of a cold-chamber die-casting machine. Metal injection should begin soon because the metal starts to solidify in the shot sleeve; and, if metal with solid particles is injected into the die, the high velocities can cause die wear and may contribute to die erosion and towards a deterioration of the quality of the castings. It is not recommended to inject immediately because it can destroy the wave pattern and can entrap air in different forms. Hence, shot-command delay is the first process parameter to be selected carefully. Then it is the first stage velocity. If it is too low and too high, it can contribute to wrong wave formation. The wave is formed if shot velocity (first-stage velocity) is too slow. The wave gets on top of the air, and the air is injected into the cavity (Thompson, 1996; Garber, 1982).

The other sleeve-related process parameters are acceleration to attain different stages of velocity and fill percentage. The acceleration can also be a deciding factor in porosity formation. Shot-sleeve percentage fill can also affect the wave formation. If the sleeve is full of metal, the air quantity is less when compared to a lesser extent of fill, and hence higher velocities can be applied safely to fill the cavity without forming deteriorated wave patterns. Plauchniak and Millage (1993) has described a four-stage shot system that adds a deceleration phase between stages in the hope to minimize impact pressure.

The process parameters affecting the entrapped air in the shot sleeve are the velocities of the plunger, shot-sleeve fill percentage and the acceleration to reach the first stage of desired velocity (Thompson, 1996). A too low first-stage velocity can form an inward wave of air entrapment in the sleeve (Figure 1). A too high velocity can form different flow in the metal towards the die cavity within shot sleeve that can result in entrapment of the air in a forward direction (Figure 2). It helps if the shot sleeve is filled more than 50% (Thompson, 1996; Walkington, 1997). It is possible to instantaneously accelerate the plunger from zero to first-stage velocity without producing porosity in 50% fill. The pressure requirements, fill time, and gate velocity very often make the 50% fill impossible (Walkington, 1997).

Garber (1982) has developed a mathematical model of the effects of plunger-related process parameters. It is noticeable that his model does not include the shot-sleeve parameter—the acceleration of plunger. In fact, in his previous work, Garber (1981)

Figure 1. Air is entrapped when shot velocity is too low in the backward reflected wave

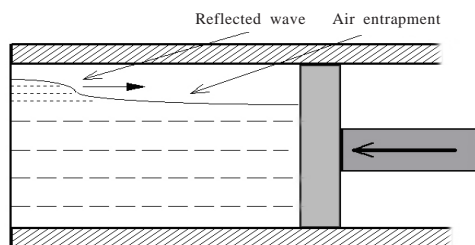
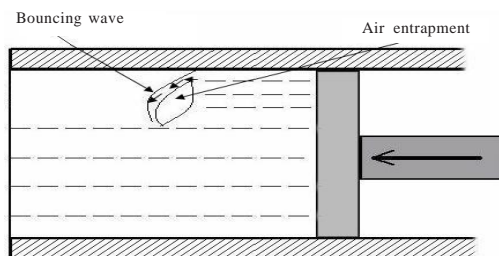


Figure 2. Air is entrapped if the shot velocity is too high; the three small arrowheads show the problematic flow responsible for entrapment



vehemently denies the importance of acceleration at all. The view that a smooth acceleration can minimize air entrapment in castings from shot sleeve, while doubtful for Garber, is considered very important by other authors (Thome and Brevick, 1995; Thompson, 1996). Thome and Brevick (1995), similar to Thompson (1996), discuss optimal acceleration to reach different stages of velocities. The authors advocate control of acceleration to reduce turbulence in shot sleeve and to minimize air entrapment for less than 50% fills. Backer and Sant (1997) found a direct effect of acceleration during a slow shot of velocity. The authors have found that high accelerations break the waves at the ends of the plunger that has the potential to entrap air while the metal is being injected in the die cavity. Slow accelerations, on the other hand, were found to be optimal in the sense that they do not break the wave and there is a low probability of air entrapment in this case. A study conducted by Brevick, Duran, and Karni (1991) addresses the issue of acceleration with respect to Garber's optimal velocities (Garber, 1982) to minimize air entrapment. It was found that an acceleration of 2 in/sec/in^1 further minimizes air entrapment at Garber's critical velocity. If acceleration is considered important, the concept of critical velocity can be applied to further low percentages of shot sleeve fills. Brevick, Duran, and Karni (1991) report achievement of nonbreaking waves up to as low a percent initial fill as 20%.

A series of work dealing with the application of control engineering to die casting emphasizes acceleration and provides the mechanism to measure and control the acceleration of the plunger for successful die casting with minimum scrap (Hadenhag, 1989; Shu & Kazuki, 1997; Vann, 1993). Vann and Shen (1989) claimed that controlled acceleration during the whole shot press (shot cycle) minimizes air entrapment and hence porosity. Hadenhag (1989) made similar claims that using controlled acceleration and deceleration gets rapid paybacks with fewer rejects, metal savings, and higher machine utilization. Similar results and conclusions have been drawn about acceleration in older die-casting literature (Kruger and Bosch, 1966; Pomplas, 1972). It seems that only Garber (1982) has disagreed with the importance of acceleration.

The velocities of first and second stages of plunger movement are other process parameters that effect the formation of porosity for pretty much the same reasons as acceleration. Both are related to the formation of "wrong" motion of liquid aluminum (waves) inside the plunger. Figures 1 and 2 show the cases with too high and a too low initial (first-stage) shot velocity (Thompson, 1996). The change over position naturally becomes important when the velocity has to be changed from first to second stage. According to Garber (1981, 1982), Thome and Brevick (1995), and Thompson (1996), porosity arises out of the suboptimal settings of parameters — namely settling time after pouring; first-stage velocity; inappropriate changeover position; and, to some extent, second-stage velocity.

Garber's pioneering paper (1982), supported with a mathematical model of porosity formation, remained the center of discussion for a decade and a half. Garber identified further two shot-sleeve-related parameters that affect air entrapment in a cold-chamber machine. They are initial fill percentage and diameter of the plunger itself.

Hairy, Hemon, and Marmier (1991) designed an expert system to diagnose and suggest solutions to the die-casting problems. According to the authors, most defects result from poor settings of machine parameters like first- and second-stage velocities and overpressure.

Asquith (1997) studies the effect of first- and second-stage plunger velocities, changeover position, intensification pressure, and biscuit length. The author observes an increase in porosity with increasing first-stage velocity with no significant effect on surface or X-ray quality test results. Second-stage velocity should be low to achieve low porosity but a higher second-stage velocity is required to minimize surface defects. It is suggested to have a 3.5 m/s second-stage velocity considering other factors like die wear and flashing that can occur with higher velocities.

Asquith (1997) and others (Andresen & Guthrie, 1989; Backer & Sant, 1997; Brevick, Duran, & Karni, 1991; Garber, 1982; Garber & Draper, 1979a, 1979b; Hadenhag, 1989; Kruger and Bosch, 1966; Plauchniak & Millage, 1993; Shu & Kazuki, 1997; Vann & Shen, 1989) have unanimous agreement that shot velocities are crucial to the quality including occurrence of porosity in high-pressure die casting. Aforementioned authors describe all their systems under two stages of velocities in sharp contrast to Plauchniak and Millage, who argue that third-, even fourth-stage velocity systems are better. The first two stages are essentially the same for elimination of gases through forming a wave that eliminates them before entering gate and runner systems. Second stage is to fill the cavity by matching the resistance offered to the flow-by runner. The third stage is to enhance solidification (intensification) pressure. The fourth-stage system described by the authors actually adds a deceleration stage between the first two stages. The authors argue that this stage breaks any spike in pressure developed when the cavity is filled and can increase die life.

The effect of the changeover position is very interesting. The porosity decreases with an increase in the changeover position. Increasing it to 600 mm produced the least porosity, and all castings passed the visual tests. Asquith (1997) does not point out the effect of a high second-stage velocity and a high changeover position. It is worth studying if it is possible to have a high second-stage velocity with a high changeover position to minimize porosity, as well as surface defects. The effects of combination of this configuration on die wear and die flashing can also be investigated.

Vents, Pressure, and Gas-Related Porosity

The air in a cavity can be entrapped due to the problems in runners or ventilation. The vents should be big enough to let the air escape and be located near the last spot to solidify. The runner should not have sharp corners in general. If the vents are working properly, the air entrapped can escape to a sufficient extent (Walkington, 1997).

The purpose of the application of high pressure in die casting is to minimize shrinkage apart from rapid production, low costs, and to achieve a lower cycle time. In HPDC, no extra metal is generally provided to reduce shrinkage porosity that is a result of volumetric contraction. Many die casters still find shrinkage-related porosity despite applying enough pressure, because the applied pressure can be different than the actual pressure developed inside cavity. This happens because of insufficient biscuit size or too big a size and unexpected solidification. If the biscuit is too small, it can solidify first or even metal in the shot sleeve can solidify which can take pressure off the cavity.

Asquith (1997) observed double porosity when he applied “no intensification pressure” (which means that a base pressure of 25.8MPa was applied). Here, the author was able

to test the plunger movement with high pressure and concluded that high intensification pressure has a more significant effect on porosity than the plunger-speed configuration. It is worthwhile here to point out that the porosities that result from velocity profiles and intensification are two entirely different kinds of porosities: gas and shrinkage porosities.

The quantity or type of lubricants used to grease the die and plunger can be a significant contributor to porosity if they get burnt and result in the formation of gas. The purpose of die lubricant is easy extraction of the part after solidification, while plunger lubricant is used to facilitate motion of the heavy plunger through the cylinder.

Due to the extreme temperatures in the die-casting environment, some of the lubricant gets burnt and produces gases. An optimal amount of lubricant that is dispersed evenly is used to reduce lubricant porosity. Water is an integral part of die lubricants, and it can occur as steam porosity due to high temperatures. Water can accumulate on a die from a sprayer and leaking water-cooling lines.

Porosity Models

Gordon, Meszaros, Naizer, and Mobley (1993) have developed a model to calculate porosity in terms like the volume of liquid in the casting cavity, which does not require extra metal supply to compensate for shrinkage, volume of cavity, temperature of the gas in the casting cavity, pressure applied to the gas during solidification, liquid alloy density at the melting temperature, solid alloy density at the melting temperature, quantity of gas contained in the casting at standard temperature and pressure (STP), solubility limit of gas in the solid at the melting point, or solidus temperature at STP. It is noticeable that some of these are not die-casting machine parameters. The authors correlate the results of other researchers in terms of die-casting process parameters like volume of die lubricant per casting, plunger lubricant (amount), state of shot sleeve, cavity fill time, fast-shot velocity, die-temperature gradient, metal temperature in the furnace, and die open time.

This work is of particular interest to the authors of this chapter because the model proposed by Gordon et al. (1993) is helpful in calculating porosity but does not provide any direct recommendations on how to reduce it, as it does not address the formation of porosity in terms of die-casting process parameters. This warrants further work to verify the model given by Gordon et al. (1993). The authors do not have a framework to fit in the die-casting process parameters in their mathematical model; however, die-casting process is essentially controlled by its process parameters. One of the observations going against the model of Gordon et al. (1993), as reported by Garber and Draper (1979a), is the decrease in porosity with the decrease in holding temperature. It is assumed that the decrease in temperature may affect the volume of liquid in the casting cavity that is not supplied with extra liquid metal (because it is not required to) to compensate for solidification shrinkage and the gas that is entrapped in the casting cavity. This is further needed to be investigated and can result in a change in the model of Gordon et al. (1993).

Significant work has been done in Australia recently with novel approaches and applications to the porosity-modeling problem (Rogers, Yang, Gershenzon, & Vandertouw, 2003). The authors put emphasis on the data-acquisition (shot-monitoring) kit. They

have developed revolutionary technology that has the ability to “look into a casting” and signal the red/green light to indicate rejects. Our work (Khan, Frayman, & Nahavandi, 2003) uses an artificial neural network (ANN) to predict porosity reliably in aluminium HPDC. The work by Huang, Callau, and Conley (1998) is a similar attempt, but it is not related to HPDC. Yarlalagadda and Chiang (1999) have used neural networks to find out the intradependence of process parameters in the die-casting process. Our work is different from the work noted previously since it is an attempt to model HPDC process defects given the process parameters, which represent the state of the machine at a given instant of time.

Biscuit Size

Very low and very high biscuit sizes generally result in higher porosities. An increase from 13 mm to 15 mm lengths dramatically decreased porosity (Asquith, 1997). It is recommended to use a minimum 25 mm biscuit length with maximum intensification for sake of passing the X-ray test for the casting, with ladle consistency being taken into account to maintain the size. It is noticeable that in Gordon et al. (1993) the authors do not attempt to relate the size of the biscuit to their equations. Further research can be conducted to relate biscuit size to equations or a new term added to the equations to take the biscuit size into account. The exploitation of a neural network can be a good idea, because it offers the utility of adding the biscuit size to the inputs of the network.

Methodology

Computational intelligence techniques that include ANNs, genetic algorithms, simulated annealing, and fuzzy logic have shown promise in many areas including industrial engineering where the use of neural networks, genetic algorithms, and fuzzy logic is quite prominent. The capability of ANNs to learn complex relationships well has made them a popular methodology for modeling the behavior of complex systems. Computationally, ANNs in their most common form of a multilayer perceptron (MLP) are distributed parallel-processing systems capable of a fault tolerant and efficient learning and are resistant to noise and disturbances. They are connectionist structures composed of nodes called neurons and arcs connecting the neurons with weights associated with the arcs. Weights are adaptable and are the main learning parameters of the network. The network learns typically by using a backpropagation learning algorithm (Rumelhart, Hinton, & Williams, 1986) that updates the weights. The network has generally three types of layers called input, output, and hidden layers. The information is presented in a preprocessed or raw format into the input layer of the network and the predictions are obtained at the output layer of the network.

MLPs are quite useful in developing an inductive model of the problem at hand with a fair accuracy when there is no mathematical/physical model available and have been used in the die-casting industry (Huang, Callau, & Conley, 1998; Yarlalagadda & Chiang, 1999;

Khan, Frayman, & Nahavandi, 2003).

A MLP can be represented symbolically as:

$$y_{r,n}^p = \sum_{\gamma=1}^{\gamma=L} \sum_{\eta=1}^{\eta=N_L} \Phi \left(\sum_{\substack{i=1 \\ j=1}}^{\substack{i=n \\ j=m}} x_i \theta_{ij} \right)$$

Here:

$y_{r,n}^p$ is a noisy output especially in the case of real-world data,

p is a pattern number,

r represents noise (random distribution),

n is the number of output components (neurons),

θ_{ij} is the adjustable parameter of the model (weight),

Φ is the transfer function, and

i and j are the neuron numbers.

An MLP was selected for this work, and the aim of the work is the understanding and modeling of the casting defects in terms of machine parameters.

Experimental Setup

An MLP with one hidden layer has been used in this work to model location and quantity of porosity in a casting. The data used to train the network consisted of process parameters related to porosity and location and quantity measures of porosity in the castings. The process parameters (the inputs to the ANN) to use were chosen on the basis of existing knowledge of the porosity-related parameters from the die-casting domain. These parameters are: first-stage velocity, second-stage velocity, changeover position, intensity of tip pressure, cavity pressure, squeeze-tip pressure, squeeze-cavity pressure, and biscuit thickness. A dataset consisting of 306 data points and obtained from data logging the operation of a multicavity HPDC machine in a die-casting manufacturing environment has been used. The first 204 data points in a dataset were used for training, and the remaining 102 points were used for testing.

The level of porosity was quantified using X-ray grades at two different locations labeled as A and E. These X-ray grades are quality measures ranging from 1 to 4, with 1 representing minimum level of porosity at the designated location and 4 representing the worst porosity level. Occurrence of porosity level of 4 in any of the locations on the casting results in the casting being rejected. The outputs of MLP are the levels of porosity (quality) measures at location A and E in the casting.

We obtained the neural network model of porosity by training the network using a backpropagation learning algorithm (Rumelhart, Hinton & Williams, 1986) with a different number of hidden neurons and selected the one that provided the best generalization on unseen test data. The best-generalized neural network obtained has four hidden neurons consisting of a sigmoid transfer function.

After modeling the die-casting process with an MLP to a sufficient degree of accuracy, we conducted conventional die-casting tests by varying one of the process parameters and keeping the others constant. This was done with a simulated process rather than on an actual die-casting machine, as experimentation on an actual die-casting machine could result in a considerable waste of resources in terms of metal, manpower, and energy and incur a significant cost. There are several types of sensitivity analyses that can be performed including the weight-based sensitivity analysis based on the change in outputs (Baba, Enbutu, & Yoda, 1990) and the sensitivity analysis based on energy functions (Wang, Jones & Partridge, 2000). We have selected the sensitivity analysis based on the changes in output with respect to a change in one of the input variables as it gives us a clear indication of which input (process parameter) is having what quantitative effect on the output (the porosity).

The sensitivity analysis is generally performed around the average values of the elements of an input vector with one element going through a change between two extremes in an operating window.

For input set X of column vectors:

$$X = [X_1, X_2, X_3, \dots, X_n]$$

Each column vector consists of further scalars:

$$X_p = [x_1, x_2, x_3, \dots, x_m]$$

We defined the average for all elements of the set X , except the element X_s that was being considered for analysis to find out the effect ΔO on the output O of the MLP model.

$$\overline{X_j} = \sum_{l=1}^m \frac{x_l}{m}$$

Here $\overline{X_j}$ is the average of the j^{th} column vector $\overline{X_j}$ in input domain set X , m is total number of scalars of which $\overline{X_j}$ is comprised of, and l is summation index.

The X_s varied between the two extremes, while all other elements were kept constant at $\overline{X_j}$. The interval of the variation of X_s was set as $[a, b]$. The variation started from a and terminated on b by an increment Δ . The data D to analyze X_s were generated initially by

$$D = a + \Delta$$

and then for the rest of iteration by

$$D = D + \Delta$$

The iteration was stopped when D had reached b . During the variation of X_s , the output is logged and plotted to visualize the change in ΔO . The general equation through which this is generated and averaged data is passed was:

$$O = net_{ho} f(D, \overline{X_j}, net_{ih})$$

Here net_{ho} are the weights associated with hidden and output layers and net_{ih} are the weights between hidden and input layers. $\overline{X_j}$ is the average of all inputs other than X_s , since D is the representative of X_s , and f is the function performed by the hidden layer of the MLP model.

Results and Discussion

The criterion that was used to judge the model quality was the agreement of the MLP model with the existing work in porosity modeling. We have found that in most cases, the MLP model of porosity formation was able to represent the underlying process well. Figure 3 shows that the obtained MLP model was able to predict accurately that the increase in first-stage velocity has a decreasing effect on the level of porosity in agreement with Garber (1981, 1982) and Thome and Brevick (1995).

Figure 4 shows that the obtained MLP model was able to predict accurately that an increase in second-stage velocity (high-speed velocity) decreases the amount of porosity in accordance with the concept of critical velocity (Garber, 1982). The porosity decreases sharply with initial increases in the high-speed velocity and then tends to stabilize as it reaches the critical velocity when it matches the resistance offered by the gate and runner system in order to inject the metal immediately as it reaches the end of the shot sleeve.

Figure 5 shows that the obtained MLP model predicts that an increase in changeover position decreases the amount of porosity. This result is in conflict with the existing work on porosity (Asquith, 1997). Further investigation is needed to determine why the MLP model has determined it in such a way.

Figure 6 shows that the increase in tip-intensification pressure is increasing the amount of porosity contrary to what should happen. Part of the problem is the pressure that is transferred inside the cavity. This result has to be seen in tandem with the next result and

Figure 3. Relationship between the level of porosity and the slow-stage velocity (also known as first-stage velocity) measured in meters per second (m/s); the Y-axis represents the quantity of porosity between levels 1 to 4, with one as minimum and four as maximum

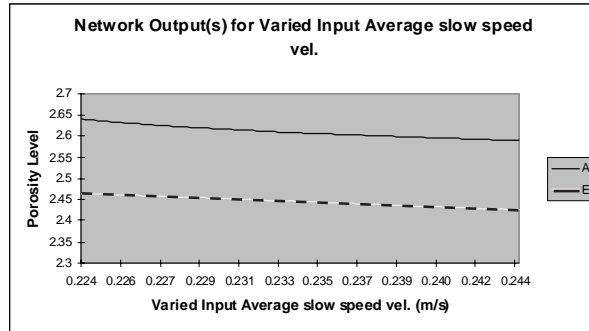


Figure 4. Relationship between the level of porosity and the high-stage velocity (also known as second-stage velocity) measured in meters per second (m/s)

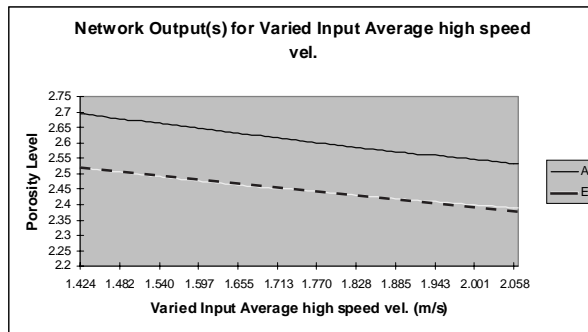


Figure 5. Relationship between the level of porosity and the changeover position (mm)

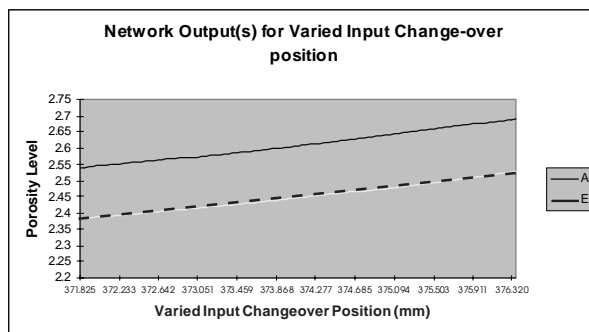


Figure 6. Relationship between the level of porosity and the intensification of tip pressure (MPa)

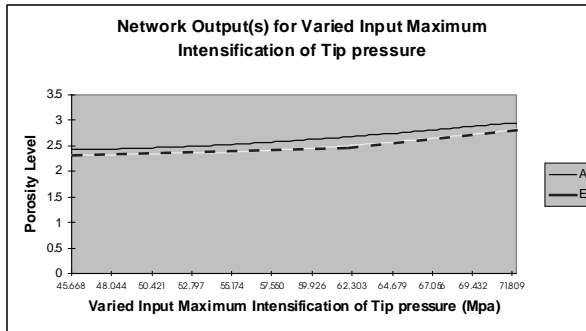
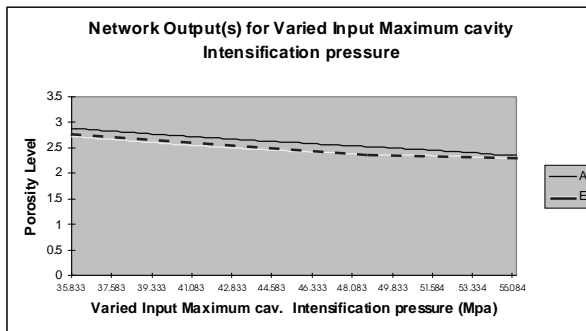


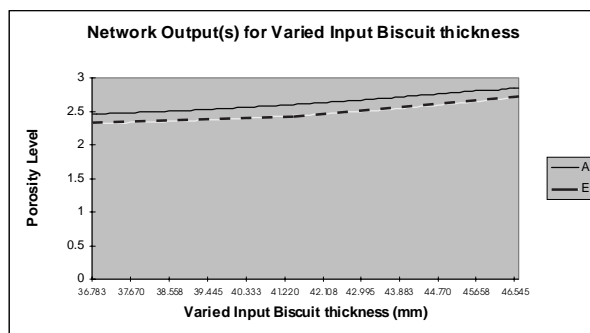
Figure 7. Relationship between the level of porosity and the maximum cavity-intensification pressure (MPa)



has shown the capability of the MLP to model the HPDC process well. Figure 7 shows that the increase in cavity-intensification pressure lowers the porosity. It is the pressure that develops inside the cavity and is a result of a tip-intensification pressure.

The porosity is supposed to decrease with increasing pressure (Kay et al., 1987; Murray et al., 1990). Figure 6 shows an increase in porosity with increasing tip pressure while Figure 7 shows a decrease with increasing cavity pressure in accordance to Kay et al. (1987) and Murray et al. (1990). That means that the MLP has been able to learn that the cavity pressure has a real decreasing effect on porosity. Applying more pressure normally reduces gas porosity. The pressure helps the gas to escape out of the casting. It is the pressure that reaches the casting rather than the applied pressure that makes the difference and the MLP has been able to predict this accurately.

Figure 8. Relationship between the level of porosity and the biscuit size (mm)



The dataset that we used had larger biscuit sizes (greater than 25 mm). The porosity is increasing with an increase in biscuit size — in accordance with the literature (Asquith, 1997). According to Asquith, a very low biscuit size (i.e., lower than 13 mm) and a biscuit size higher than 25 mm further increases porosity. In our dataset, the biscuit sizes happen to be higher than 25 mm.

Future Work

We have recently developed a novel type of MLP — a Mixed Transfer Function Artificial Neural Network (MTFANN), customized to obtain insights into data domain from the developed MLP model (Khan, Frayman & Nahavandi, 2004). The novel MLP contains more than one type of transfer function that simplifies the process of knowledge extraction from an MLP model. The application of the MTFANN to HPDC process-monitoring data is on our future agenda to provide further insights into the HPDC process.

In this chapter, we have followed the classical approach used by the die-casting industry to vary a process parameter in order to discover its effect on the quality of output (casting). The industry has a limitation of resources that prevents it from further increasing the combinatorial complexity of experiments. We can however change more than one process parameter at the same time using ANNs to study combinatorial effects of the inputs (process parameters) on the output.

Conclusion

The developed neural network model of the presented work is able to model the complex realities of HPDC. Several previous attempts in the field to model porosity using simpler

methods produce contradictory results. We believe that the usage of simpler methods is the main reason that there has not been much consensus in the work on HPDC modeling. If advanced computational intelligence techniques, such as neural networks, are further used and receive favorable response from material scientists, then it is a possibility that some sort of consensus can be obtained.

Acknowledgments

Special thanks are due to Mr. Gary Savage of CAST, Nissan, and CSIRO for providing data. The funding for the project is provided by Co-operative Research Center for CAST metals, manufacturing under the grant number AD0281.

References

- Andresen, W. T., & Guthrie, B. (1989). Using Taguchi and meltflow to determine relationships between process variables and porosity (Transaction No. G-T89-082). In *Transactions of the 15th International Die Casting Congress and Exposition, NADCA*.
- Asquith, B. M. (1997). The use of process monitoring to minimize scrap in the die casting process (Transaction No. M-T97-063). In *Transactions of the International Die Casting Congress, NADCA*.
- Baba, K., Enbutu, I., & Yoda, M. (1990). Explicit representation of knowledge acquired from plant historical data using neural network. In *Proceedings of the International Conference on Neural Networks* (pp. 155-160). New York: IEEE.
- Backer, G., & Sant, F. (1997). Using finite element simulation for the development of shot sleeve velocity profiles (Transaction No. M-T97-014). In *Transactions of the International Die Casting Congress, NADCA*.
- Brevick, J. R., Duran, M., & Karni, Y. (1991). Experimental determination of slow shot velocity-position profile to minimize air entrapment (Transaction No. D-T91-OC4). In *Transactions of the 16th International Die Casting Congress, NADCA*.
- Davis, A. J. (1978, March-April). Graphical method of analyzing and controlling hot chamber die casting process. *Die Casting Engineer* (2).
- Garber, L. W. (1981, July-August). Filling of cold chamber during slow-shot travel. *Die Casting Engineer* (4).
- Garber, L. W. (1982, May-June). Theoretical analysis and experimental observation of air entrapment during cold chamber filling. *Die Casting Engineer* (3), 14-22.
- Garber, L. W., & Draper, A. B. (1979a). The effects of process variables on the internal quality of aluminium die castings (Transaction No. G-T79-022). In *Transactions of 10th SDCE International Die Casting Congress, NADCA*.

- Garber, L. W., & Draper, A. B. (1979b, November-December). Shrinkage in #380A aluminium alloy. *Die Casting Engineer* (6).
- Gordon, A., Meszaros, G., Naizer, J., & Mobley, C. (1993). Equations for predicting the percent porosity in die castings (Transaction No. C-T93-024). In *Transactions of the 17th International Die Casting Congress, NADCA*.
- Hadenhag, J. G. (1989). Real time closed-loop control system for the shot end (Transaction No. G-T89-022). In *Transactions of the 15th International Die Casting Congress and Exposition, NADCA*.
- Hairy, P., Hemon, Y., & Marmier, J. Y. (1991). Diadem — An expert system for pressure die casting (Transaction No. G-T91-053). In *Transactions of the 15th International Die Casting Congress and Exposition, NADCA*.
- Huang, J., Callau, P., & Conley, J. G. (1998, June). A study of neural networks for porosity prediction in aluminium alloy A356 castings. In B. G. Thomas & C. Beckermann (Eds.), *Modelling of casting, welding, and solidification processes, VIII, TMS* (pp. 1111-1118).
- Jain, A. S., & Meeran, S. (1998). *A state-of-the-art review of job-shop scheduling techniques* (Technical Report). Dundee, Scotland: University of Dundee, Department of Applied Physics, Electronic and Mechanical Engineering.
- Kaghatil, S. S. (1987). Effects of silicon content on porosity in aluminium die castings with more than — inches wall thickness (Transaction No. T-T87-012). In *SDCE 14th International Congress and Exposition, NADCA*.
- Kay, A., Wollenburg, A., Brevick, J., & Wronowicz, J. (1987). The effect of solidification pressure on the porosity distribution in a large aluminium die casting (Transaction No. M-T97-094). In *Transactions of The International Die Casting Congress, NADCA*.
- Khan, M. I., Frayman, Y., & Nahavandi, S. (2003). Improving the quality of die casting by using artificial neural network for porosity defect modelling. In *Proceedings of the 1st International Light Metals Technology Conference 2003* (pp. 243-245). Australia: CAST Centre Pty Ltd.
- Khan, M. I., Frayman, Y., & Nahavandi, S. (2004). Knowledge extraction from a mixed transfer function artificial neural network. In *Tech'04 Proceedings of the Fifth International Conference on Intelligent Technologies 2004*. Houston, TX: University of Houston, Downtown.
- Kruger, G. H., & Bosch, R. (1966). Oscillographic studies of plunger velocity and metal pressure relationships during and after the filling of a die. In *4th International Die Casting Exposition and Congress* (Paper 1304).
- LaVelle, D. L. (1962). Aluminium die casting and factors affecting pressure tightness. *Transactions of American Foundrymen's Society*, 70, 641-647.
- Murray, M. T., Chadwick, G. A., & Ghomashchi, M. R. (1990, June). Aluminium alloy solidification in high pressure die casting. *Materials Australasia*, 22(5), 20-21.
- Plauchniak, M., & Millage, B. A. (1993, November-December). New closed shot control system features total integration. *Die Casting Engineer* (6).

- Pomplas, L. J. (1972). Digital servo control of shot plunger velocity and acceleration. In *Transactions of the 7th International Die Casting Congress and Exposition* (Paper P1572).
- Rogers, K., Yang, S., Gershenzon, M., & Vandertouw, J. (2003, October). Detecting porosity rejects in pressure die castings from shot monitoring. In *Proceedings 9th International Conference on Manufacturing Excellence (ICME'03)*, Melbourne, Australia.
- Rumelhart, D., Hinton, G., & Williams R. (1986). Learning internal representations by error propagation. In D. Rumelhart et al. (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations* (pp. 318-362). MIT Press.
- Shu, S., & Kazuki, H. (1997). Direct drive closed loop shot control (Transaction No. M-T97-062). In *Transactions of the International Die Casting Congress, NADCA*.
- Thome, M., & Brevick, J. R. Optimal slow shot velocity profiles for cold chamber die casting (Transaction No. I-T95-024). In *Transactions of the 18th International Die Casting Congress, NADCA*.
- Thompson, S. (1996). *Mechanisms of leaker formation in aluminium high pressure die casting*. Unpublished master's thesis, The University of Queensland, Department of Mining and Metallurgical Engineering, Australia.
- Vann, J. R. (1993). Real time dynamic shot control for improved quality and productivity (Transaction No. C93-125). In *Transactions of the 17th International Die Casting Congress, NADCA*.
- Vann, & Shen (1989). Computer integrated information and control systems for improved quality and profit (Paper No. G-T89-021). In *Transactions of the 15TH International Die Casting Congress and Exposition*.
- Walkington, W. G. (1990, September-October). Current die casting research projects conducted at Ohio State University. *Die Casting Engineer*.
- Walkington, W.G. (1997). *Die casting defects*. Rosemont, IL: NADCA.
- Wang, W., Jones, P., & Partridge, D. (2000). Assessing the impact of input features in a feed-forward neural network. *Neural Computing and Applications*, 9, 101-112.
- Yarlagadda, P. K. D. V, & Chiang, E. C. W. (1999). A neural network system for the prediction of process parameters in pressure die casting. *Journal of Materials Processing Technology*, 89-90, 583-590.

Endnote

¹ The acceleration is measured in L/T/L dimensions rather than L/T/T in die casting.

Chapter XII

Neural Network Models for the Estimation of Product Costs: An Application in the Automotive Industry

Sergio Cavalieri, Università degli Studi di Bergamo, Italy

Paolo Maccarrone, Politecnico di Milano, Italy

Roberto Pinto, Università degli Studi di Bergamo, Italy

Abstract

The estimation of the production cost per unit of a product during its design phase can be extremely difficult, especially if information about previous similar products is missing. On the other hand, most of the costs that will be sustained during the production activity are implicitly determined mainly in the design phase, depending on the choice of characteristics and performance of the new product. Hence, the earlier the information about costs becomes available, the better the trade-off between costs and product performances can be managed. These considerations have led to the development of different design rules and techniques, such as Design to Cost, which

aims at helping designers and engineers understand the impact of their alternative decisions on the final cost of the developing product. Other approaches, which are based on information about already designed and industrialised products, aim at correlating the product cost with the product's specific characteristics. The real challenging task is to determine such a correlation function that is generally quite difficult. The previous observation led the authors to believe that an artificial neural network (ANN) could be the best tool to determine the correlation between a product's cost and its characteristics. Several authors hold that an ANN can be seen as a universal regressor, able to approximate any kind of function within a desirable range, without the necessity to impose any kind of hypothesis a priori on the characteristics of the correlation function. Indeed, test results seem to confirm the validity of the neural network approach in this application field.

Introduction

The ever-growing competitive pressures that characterise most industry sectors force firms to develop business strategies based on a large number of differentiation factors: higher quality and service levels, as well as customisation and continuous innovation.

The research of organisational, technological, and managerial solutions and tools that can shift the trade-off line between costs and differentiation is extremely important in a competitive environment. In this perspective, the “process view” has been given great attention in all managerial and organisational disciplines, and the development of the theory of “management by processes” has led to the gradual elimination of functional barriers. In addition to being responsible for the results of his or her unit, each functional manager is usually in charge of the overall effectiveness of the processes in which his or her unit is involved, following to a typical input-output logic (Berliner & Brimson, 1988; Hammer & Stanton, 1999; Zeleny, 1988).

Obviously, this process reorientation requires the implementation of a radical cultural change supported by a tailor-made re-engineering of the organisational structure and of the management-control systems, with particular regard to performance-measurement systems.

In particular, the R&D department is one of the areas most involved in the process of organisational change. Since the R&D unit is mainly made up of technical or scientific experts, during the new product development (NPD) process this unit traditionally puts much more emphasis on the technologically innovative contents and on the absolute performance of the product, rather than on the impact of the adopted solutions on convenience and results (like the manufacturing costs or the contribution margin generated by the new product).

The estimation of future product costs is a key factor in determining the overall performance of an NPD process; the earlier the information is known, the better the relationship between costs and product performances will be managed. Typically, the cost per unit of a given finished product is the sum of different kinds of resources — raw

materials, components, energy, machinery, plants, and so on — and the quantification of the usage of each resource is extremely difficult in the first stages of the life cycle (and particularly in the early phases of the product-development process), given the reduced amount of information and the low level of definition of the project.

All these considerations justify the effort made in the development of techniques and approaches to cope with the problem of estimation of manufacturing costs in highly uncertain contexts. This chapter illustrates the results of a study aimed at comparing the results of the application of two techniques: the parametric approach (perhaps one of the most diffused in practice) and a predictive model based on the neural network theory.

Six sections follow the introductory section. The following one deals with the strategic relevance of cost management in modern competitive scenarios. Then, authors provide a brief review of the main cost-estimation approaches found in scientific literature, while the fourth section illustrates the basic theoretical elements of ANNs. The following three sections illustrate a case study first describing the application context (the characteristics of the firm, of the product, and of production technologies), and then illustrating the design, development, and refinement phases of the two models, as well as the comparison of results. The last section is devoted to the conclusions.

The Strategic Relevance of the Cost Variable in the New-Product-Development Process

The process view of the firm can be of great help in making designers and product engineers more aware of the critical role played in determining the overall economic performance of a firm, as proved by the “life-cycle-costing” theory (Blanchard, 1979; Fabrycky, 1991; Shields & Young, 1991).

The life-cycle-costing theory states that, although the great majority of costs of a finished good are generated in the manufacturing/distribution stage (given also the repetitive nature of these activities for almost all kinds of products), most of these costs are implicitly determined in the early phases of development. Figure 1 shows the difference between the “actual costs” and the “committed costs” curves. The latter one is built by “translating” the costs incurred in the various stages of the life cycle back to the instant in which the different decisional processes that implicitly fixed those costs took place.

These considerations have led to the development of a group of techniques, whose objective is to help engineers and designers in their decisional processes and make them aware of the implications of the alternative design solutions on the future costs of the product (Ulrich & Eppinger, 1995).

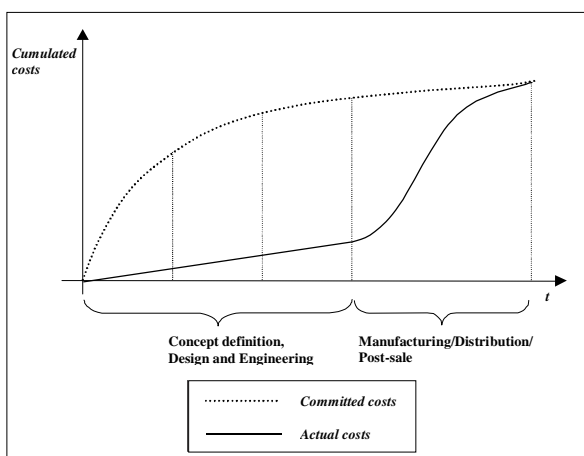
The approaches are named after their specific application context and their emphasis on economic figures (Huang, 1996), but two main types can be identified:

- **Design for “X” family** (which, in turn, includes Design for Manufacturing/ Assembly, Design for Inspection and Testing, Design for Maintenance, etc.): design rules finalised mainly to the reduction of costs through the standardisation of components and simplification of subsequent manufacturing and product-maintenance processes;
- **Design to Cost:** if compared to the “Design for X” approaches, a more structured and analytical approach, whose aim is the quantification of the economic impact of the different design solutions adopted.¹

The strategic relevance of future production costs has lead to the development of a new, rather revolutionary, approach to the management of the new product-development process: *Target Costing* (Ansari, Bell, & the CAM-I Target Cost Core Group, 1997; Cooper, 1997; Hiromoto, 1988; Sakurai, 1989).

While in the “traditional” process the economic/financial justification (the investment appraisal) is conducted only in the last phases of the NPD process (after the design or even during the engineering phase), according to the Target Costing philosophy (particularly diffused in the Japanese automotive industry) the starting point is the determination of the estimated market price of the new product, given the estimated release time and the target market segment. Such information, coupled with the expected (desired) profitability margin, leads to the identification of the sustainable production cost per unit. All the subsequent design and development phases must then be “cost” driven (i.e., all decisions must be made according to the final objective of meeting the target production cost). The overall production cost is then divided into two main parts: external and internal costs. These are then split into “third level” costs (respectively, one for each purchased part or acquired service, and one for each main internal process/

Figure 1. Committed costs and actual costs along the life cycle of a product



activity). Consequently, these costs become the fundamental reference elements in all the subsequent steps of the NPD process; all decisional processes must take into consideration the effects on these figures. Indeed, the overall estimated production cost must not be higher than the initial target cost.

All the aforementioned managerial approaches highlight the strategic relevance of the information regarding the future manufacturing cost of the product (or of its components). Indeed, in the life-cycle theory, the overall objective resides on the minimisation of the (expected) cumulated life-cycle cost. Hence, the first step is the estimation of costs in each phase of the life cycle. Manufacturing costs usually represent the most relevant component.

Similarly, in a firm which adopts the target-cost approach, the accurate estimation of future manufacturing costs is fundamental to understand whether the overall target cost can really be reached or not. Moreover, if an assembler firm can make reliable predictions about the production costs of its suppliers (for purchased components), its bargaining power will be higher due to the reduction of information asymmetry (Porter, 1980). This appears particularly critical in the target-cost approach, due to the “pressure” that is made on suppliers to meet the objective.

The next section is devoted to the illustration of the state-of-the-art on the cost-estimation issue, with particular regard to the latest developments.

The Cost-Estimating Techniques

In literature, three main quantitative cost-estimation approaches can be identified:

- *Analogy-based techniques:* These techniques belong to the category of qualitative estimation methods. They are based on the definition and analysis of the degree of similarity between the new product and another one already produced by the firm. The underlying concept is to derive the estimation from actual information regarding real products. However, many problems exist in the application of this approach, due to:
 - The difficulties faced in the operationalisation of the concept of “degree of similarity” (how to measure it?); and
 - The difficulty of incorporating the effect of technological progress and context factors in the parameter.
- This kind of technique is mainly adopted in the first phase of the development process of a product because it allows obtaining a rough but reliable estimation of the future costs involved.

- *Parametric models:* According to these quantitative techniques, the cost is expressed as an analytical function of a set of variables. These variables usually consist in some features of the product (product performances, morphological characteristics, type of materials used), which are assumed to have a major impact on the final cost of the product (for this reason, they are generally named “cost drivers”). These analytical functions are usually named Cost Estimation Relationships (CER), and are built through the application of statistical methodologies, such as regression techniques (e.g., see NASA, 1996). They can be adopted during the development of new products and as a control during the implementation, providing a target for the final cost of the product. Although they are mainly used for the estimation of the cost of large projects (such as in the aeronautical field), they could be effective also for the estimation of the cost of those products, where the cost drivers could be easily identified.
- *Engineering approaches:* Following this approach, the estimation is based on the detailed analysis of the features of the product and of its manufacturing process. The estimated cost of the product is calculated in a detailed analytical way, as the sum of its elementary components, constituted by the value of the resources used in each step of the production process (e.g., raw materials, components, labour, equipment, etc.). The result is the so-called “standard cost” of the product.

Moving from the first to the last, the average precision of the methodology increases (obviously along with its cost). However, the choice between the three methodologies is not completely free. Each one of them suits to different stages of the NPD process, given their different degree of analyticity and the different amount of input data needed. While the analogy-based models can be implemented already in the early stages of the process, the engineering approach can be used only when all the characteristics of the production process and of the product are well defined (i.e., at the end of the engineering phase).

Within these methods, the parametric model represents a good trade-off between precision and effort required in the model definition, especially when regression techniques are used in order to identify the CER (i.e., to correlate input factors — characteristics of the products — to output factors — the product’s performances). Indeed, the regression technique has well-known mathematical basis and has proven to be a robust model in many situations.

For this reason, in the remainder of the chapter, a parametric model is assumed as a basis for the comparison of the performances of the ANN.

Artificial Neural Networks for Cost Estimation

Literature Review of Manufacturing Applications of ANNs

Although the neural network theory has been applied in most disparate sectors, this approach was first used in the manufacturing sector for planning, emulation, and management of production processes and plants. For example, Cavalieri and Taisch (1996), Cavalieri, Garetti, and Taisch (1995), and Cavalieri, Rabe, and Taisch (1997) have developed ANNs for the design of hybrid intelligent systems and of process plants.

Many other works are devoted to the application of ANNs to forecasting problems, such as those of Hill, O'Connor, and Remus (1996) — in which ANNs are compared with other statistical forecasting methods, showing better performances — and O'Rourke (1998), which deals with neural networks for the estimation of the market value of equity stocks, and concludes that the results achieved are better than those of a linear predictive model.

In the field of cost estimation, the work of Shtub and Zimerman (1993) should be mentioned, which presents the results of the application of ANNs vs. regression models for the determination of product costs in assembly industries.

In the specific case reported in the present chapter, the use of ANNs for product-cost estimation represents a relatively new approach grown in popularity in the last years, although a little amount of research seems to exist in this field at present.

One of the most significant works is that of Zhang, Fuh, and Chan (1996), which illustrates the use of a neural-network-based model for the estimation of the packaging cost, based on the geometrical characteristics of the packaged product (the so-called *feature-based cost*). The paper states that the elements impacting the final cost of the product can be classified into two subsets: (1) *explicit cost drivers* (such as material cost) obtained directly; and (2) *implicit cost drivers* (such as the relationship between explicit cost drivers) inferred by historical data. In other words, while explicit cost drivers are the basic elements of cost, the implicit drivers represent the function that links those elements in order to obtain the final cost of the product.

The ANNs, in this context, are suitable for the identification of such implicit cost drivers. Based on the analysis of the literature and the research conducted, the main benefits of the ANNs for the cost estimation can be summarised as follows:

- **Nonlinearity:** Neurons are *nonlinear processors*, thus, the output provided is a nonlinear function of the input. As a consequence, the network itself is a nonlinear architecture. Due to this characteristic, the ANN could cope with nonlinear data and environment.

- **Autofitting transfer function:** One of the most important applications of ANNs is the modelling of a system with an unknown input-output relationship; through the learning process, ANNs can infer such a relationship without any *a priori* hypothesis. In this sense, ANNs are often referred to as *universal regression systems* (Hornik, Stinchcombe, & White 1989).
- **Adaptivity:** ANNs have the ability to adapt their response to the changes in the surrounding environment. Indeed, an ANN can be retrained if the environment changes substantially.
- **Fault tolerance with respect to the data:** ANNs can provide good response even if the input data are not completely correct.

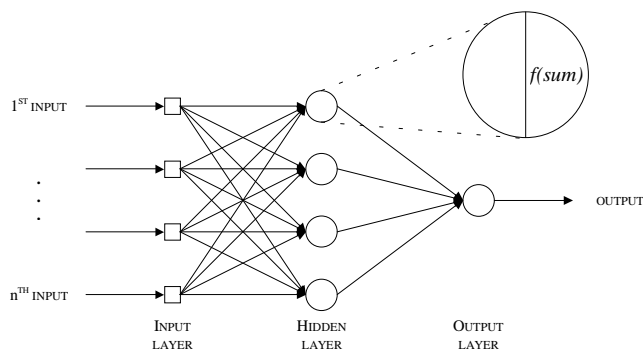
With regard to the architecture of ANNs, in literature there is a multitude of architectures and several classification frameworks. As an example, Chester (1993) classifies ANNs according to the learning method or to the organisation of the neurons.

In the present work, the ANN used is a Multilayer Perceptron (MLP) with backpropagation learning algorithm, in which neurons are organised in different layers. Each neuron has a specific function: the first one is the input layer (fed by input data), while the last one is the output layer (which provides the answer of the network). Between input and output layers there could be several other hidden layers (see Figure 2). The number of hidden layers has an important role in determining the generalisation ability of the MLP (e.g., see Lawrence, Giles, and Tsoi [1996]).

As shown in Figure 2, the general structure of a neuron is composed of two main components: the first component is the *summation operator* (Σ) which sums up the weighted impulses coming from the neurons belonging to the previous layer. The second component is the *propagation operator*, which applies the *transfer function* f to the results of the summation operator and propagates the outcome to the following layer.

The reason behind the choice of the MLP resides on its main nature of “*universal regression tools*” (Hornik et al., 1989; Mason & Smith, 1997). Such tools allow for the

Figure 2. Structure of a multilayer neural network



identification of relationships between different data sets, even if the form of these relationships is not defined *ex ante*. In particular, the work by Mason and Smith (1997) compares the performance of regression and neural network approaches for cost estimation purposes. The results show that the ANN-based models are more precise, especially when the analytical expression that links input and output variables is unknown, or when it cannot be expressed in a polynomial form.

Case Study

In order to provide an example of the application of the neural network tools in cost estimation, the following section describes an industrial case study. The objective of the research was to compare the results achieved with the application of a traditional cost-estimation technique — with particular emphasis on multiple linear regression — with those obtained through the design and implementation of an *ad hoc* MLP. The comparative analysis is especially interesting because of the lack of literature on the topic.

The analysis was conducted through a real case study provided by an industrial company operating in the automotive sector, whose main mission is the design, production, and sale of braking components and integrated systems for civil and industrial vehicles.

The customers are mainly very large industrial contractors (the major automotive multinational companies) that faced the general crisis of the sector in the last decade. The competitiveness of the automotive component sector forces firms to search for differentiation elements. At the same time, due to the relatively poor bargaining power (with respect to large customers) great attention is paid to price levels. Hence, one of the strategic objectives that has been identified by the company's top management is the analysis and reduction of product costs (both the purchasing costs of components and the internal production and assembly costs).

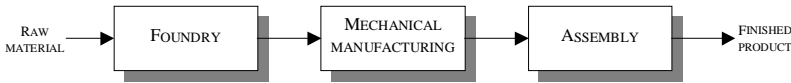
In this context, the adoption of formal methodologies for the accurate estimation of manufacturing costs of new products has been considered very important in order to pursue the claimed strategic objective of the company.

The case study focuses on the estimation of the production costs of cast-iron disk brakes, which are then assembled with other components to produce braking systems, or can be also sold directly in the spare parts market.

The overall production system of this typology of disk brakes is sketched out in Figure 3; three main production phases can be identified:

1. Foundry, which produces raw disks starting from cast iron;
2. Mechanical manufacturing, which produces the finished disks with the dimensional and surface features as specified in the project specifications; and
3. Assembly, which realises the final braking system assembling the disk with other components.

Figure 3. Main phases of disk-brake production process



Each phase has characteristics and peculiarities that contribute to the creation of a valuable product for the customers. In this context, the cost-estimation modelling techniques can be used to quantify a reference cost value for the raw disk and the finished disk as well.

In the following, for the purpose of this chapter, we focused our attention on the first phase of the production process (foundry).

The Design of the Parametric and Neural Models

As stated previously, given the objectives and specific application context of the cost-estimation methodology in the analysed firm, the performance of an ANN has been compared to the performance of another cost-estimation technique. Among the “classical” methodologies, the *parametric model* has been chosen.

As stated in the section “Cost-Estimation Techniques”, the cost of a product in a parametric model can be expressed as an analytical function of a set of *variables* that represents the features of the product (e.g., performance, morphological characteristics, type of materials used, etc.). This analytical function, also called CER, is built through the application of statistical methodologies and the linear form of such a function is used most of the time due to its simplicity. Thus, given a set of N variables (or features) $v_1 \dots v_N$, the total cost TC of the product could be expressed as:

$$TC = \alpha + \beta_1 v_1 + \beta_2 v_2 + \dots + \beta_N v_N \quad (1)$$

where α, β_i are the parameters of the function and are derived through the application of the multiple-regression technique.

This approach has a well-known mathematical basis and is easily applicable, once a consistent set of numerical data is available. On the other hand, the limit of the method is that it always considers the relationship that connects the variables linear, which is not always a realistic hypothesis.

The process followed in the development of the two models for cost estimation could be schematised as follow: (1) problem definition and data collection, (2) data analysis and model definitions, (3) model testing, (4) model comparison.

Each phase is described hereafter.

Problem Definition and Data Collection

Although this first phase could appear trivial, it is critical for the success of the subsequent activities. The aim of this phase is the definition of the project's objectives and constraints in order to clearly identify the elements that are supposed to be kept into consideration in the following analysis phase.

Moreover, the first phase is quite important in a complex company, where all the main actors of the product design and production processes must accept and share all objectives, strategies, and activities to be implemented.

Once the problem constraints and the methodology to be used have been defined, it is necessary to proceed to the collection of product data used to perform the analysis. Data collection also includes the identification of information sources and the corresponding business functions responsible for their maintenance and update.

With regard to the foundry phase of the production process, the main types of data are:

- *Technological data*, related to the production processes (i.e., type of operations performed on the product)
- *Design data*, related to the morphological features of the product (i.e., physical dimension and materials used)
- *Cost data*, such as labour costs, raw-material costs, final-product costs, and so on

Collected data represent the basis of the estimation models.

Data Analysis and Models Definition

Once the data-collection phase is terminated, the analysis of the data allows for the identification of the most meaningful cost drivers that are to be kept into account in the model's definition. One of the main characteristics of these drivers is that their calculation should be relatively simple even in the preliminary phases of the product-development process, when only first raw design specifications are assessed.

The selection of these drivers is based on the analysis of the production processes and on the identification of all the elements that play a role in such a process.

For example, in the case of a cast-iron raw product, it is clear that the weight of the product is relevant in order to define the final cost — the heavier the product, the more the necessary raw material.

On the other hand, other drivers are not so obvious. For example, the diameter of the raw disk could appear to be irrelevant if we consider only the product. But if we consider also the process, the diameter becomes relevant, since it affects the number of raw disks that could be cast in a single pouring — the larger the diameter of the disk and the fewer the number of meltable disks, the longer the process and the higher the costs.

Table 1. List of the identified cost drivers

COST DRIVER
Raw-disk weight
Type of raw material
Number and type of foundry cores
Type of disk
External diameter

Once the product cost drivers have been defined, it is necessary to evaluate the consistency of the available data in terms of measurability, reliability, and completeness (i.e., real information content). In particular, with regard to the last point, data could result unsuitable or insufficient leading to recycles on the previous phases or they could be redundant causing inefficiencies. The identified drivers are reported in Table 1.

Once the statistical consistency of the sample set of data has been tested, statistical and linear regression models are used to find out the relationship between each of the selected cost drivers and the dependent variable (i.e., the product cost).

Design of the Parametric Model

A parametric model expresses the relationship within the output variable (i.e., the cost of the raw disk) and the cost drivers (reported in Table 1 through an algebraic expression. If the expression is in the form of Equation 1, the model is linear and the parameter a and b_1 could be easily derived using the method of least squares.

Hence, using the well-assessed theory of multiple linear regression, it is possible to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to the observed data.

The major drawback of this method is that often the real relationship between different variables is not linear, so the linear regression could provide only an approximation of the real relation.

For example, Figure 4 illustrates the relation between the design weight of the raw disks (on the X axis) and their final cost (on the Y axis).

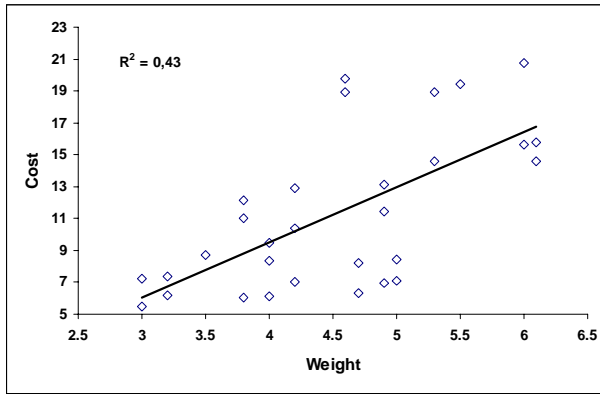
Equation 2 expresses the regression line in Figure 4:

$$Cost = \alpha + \beta \cdot Weight \quad (2)$$

where a and b have been obtained through the method of least squares.

Both the graph and the analysis of the coefficient R^2 makes it evident that such a model is not really meaningful (coefficient R^2 is expected to be closer to 1 for a good model); thus, it is possible to conclude that the weight has a certain relation with the final cost but it is not enough to explain the overall cost.

Figure 4. Approximated linear relationship between the weight and the cost of the raw disks



Hence, the simple regression model so far illustrated must be complemented with other parameters to improve its performance.

In order to reduce the error of the model, more cost drivers from Table 1 are added to Equation 2, and Equation 3 is obtained:

$$Cost = \alpha + \beta_1 \cdot Weight + \beta_2 \cdot NumCores + \beta_3 \cdot Diameter + \sum_{i \in C} \beta_i \cdot CastIron_i + \sum_{j \in T} \beta_j \cdot DiskType_j \tag{3}$$

where:

- $CastIron_i$ is a binary variable indicating the type of raw material used for the disk; there are six types of raw material that compose the set C ; obviously, for each disk only one of these variables could be set to 1.
- $DiskType_j$ is a binary variable indicating the two types of disks, normal and special, that compose the set T ; also in this case, for each disk only one of these variables could be set to 1.

Equation 3 states that the cost of the raw disk depends on:

- Factor β_1 related to the weight of the disk,
- Factor β_2 related to the number of foundry cores requested, and

- Factor β_3 , related to the external diameter.

In addition, there are three further elements that contribute to the total cost:

- The type of the disk (by a factor β_7),
- The type of raw material (by a factor β_8), and
- A residual factor α .

After its validation, the model is applied to the sample used for the estimation of the parameters, and a mean absolute percentage error (MAPE, Equation (4)) of about 9-10% is obtained. The result is a relatively good response.

Due to the fact that the parametric model assumed so far is linear, it is reasonable to suppose that the application of a nonlinear model, like an ANN, could provide even better results.

$$MAPE = \frac{1}{N} \sum_{i=1}^N \frac{|EstimatedCost_i - ActualCost_i|}{ActualCost_i} \cdot 100 \quad (4)$$

Design and Training of the Artificial Neural Network

In the discussed case, an ANN represents a valid tool for the identification of the transfer function of the analysed process, through an implicit link between the input value (the morphological and technological characteristics of the disk) and the output value (the cost).

Table 2. Input of the neural-network model

INPUT NUMBER	DATA	TYPE	RANGE
1	Disk weight	Real	(3 ÷ 6)
2	Number of cores	Integer	(0 ÷ 2)
3	External diameter	Real	(240 ÷ 255)
4	Cast Iron type A	Binary	{0, 1}
5	Cast Iron type B	Binary	{0, 1}
6	Cast Iron type C	Binary	{0, 1}
7	Cast Iron type D	Binary	{0, 1}
8	Cast Iron type E	Binary	{0, 1}
9	Cast Iron type F	Binary	{0, 1}
10	Disk type normal	Binary	{0, 1}
11	Disk type special	Binary	{0, 1}

Table 3. Results of the neural-network models

Network	Neurons in hidden layer 1	Neurons in hidden layer 2
1	5	0
2	6	0
3	7	0
4	8	0
5	9	0
6	10	0
7	5	3
8	6	3
9	7	3
10	5	5
11	6	5
12	7	5

With regard to the specific neural architecture used, it could be noted that the analysed problem could be traced back to a nonlinear regression problem, as explained in the previous section. Due to this consideration, the multilayer perceptron (MLP) with backpropagation has been preferred, since it usually leads to the most satisfactory results.

Regarding the structure of the network, the input layer is composed of 11 neurons, as explained in Table 2. The output layer is composed of only one neuron, which provides the response of the network (i.e., the cost).

The input neurons from the 4th to the 11th represent the on-off characteristics, that is, a disk could be made of only one type of cast iron and could be, of course, special or normal.

The definition of the internal structure of the ANN (i.e., number of hidden layers, number of neurons per layer, type of activation function) is generally a trial-and-error process, since no analytical procedure exists for determining the correct number of neurons and layers. After having tested more ANN configurations with different numbers of hidden layers, different numbers of neurons for each level, and different activation functions (mainly linear and sigmoid functions), the proper structure has been selected. Table 3 illustrates the ANN structures that have been tested.

More learning algorithms, such as the Levenberg-Marquardt algorithm — suitable in the case of few samples in the training set and moderate-sized ANNs — and the Resilient backpropagation algorithm, have been also experimented.

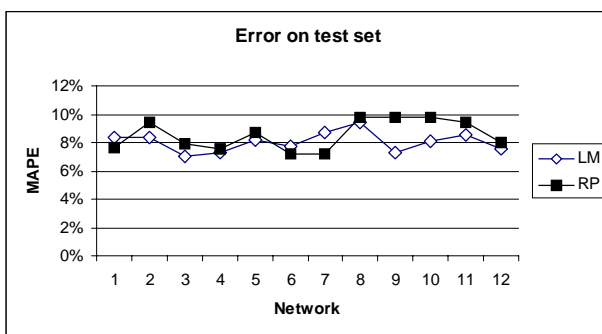
Models Testing

For the testing of both models, the set of samples has been divided into three subsets: (1) the first one, composed of 40 samples, has been used as a training set (in order to adjust the weight of the connections and to store the knowledge); (2) the second one, composed of 10 samples, has been used as a validation set, in order to avoid the network overfitting

Figure 5. Estimation error on the training set



Figure 6. Estimation error on the test set



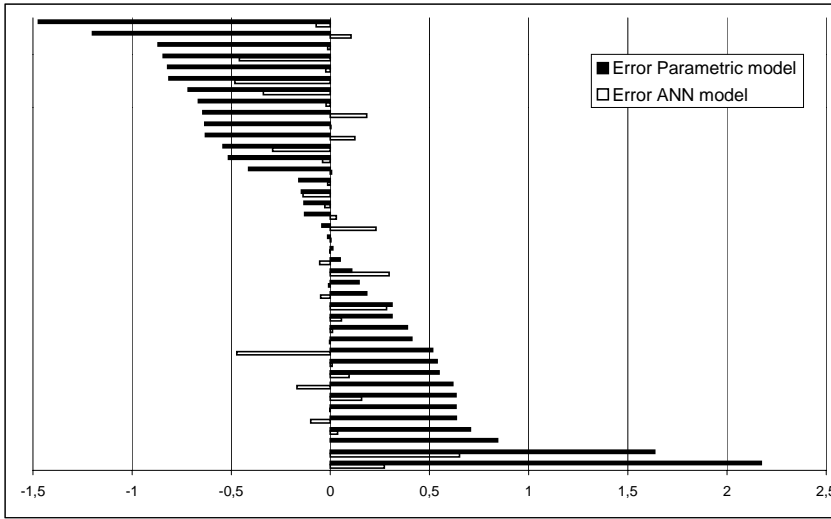
problem (that occurs when the network fits very accurately the training data while results in a poor generalisation on out-of-sample data) during the learning phase, applying the early stopping procedure; (3) the third one, composed of 10 samples, has been used as a test set to evaluate the responses of the net on unseen data (in order to evaluate the degree of generalisation).

The parametric and ANN models have been tested by comparing the model results and the actual costs of the training set used for the definition of the parameters and for the training of the ANN. The results refer to the Levenberg-Marquardt (LM) and to a Resilient Backpropagation (RP) learning algorithm.

In Figure 5 and 6, the performances of the models, measured through the MAPE indicator, are reported.

It is evident that the one- and two-layer configurations show almost the same performance, especially using the LM learning algorithm.

Figure 7. Neural network and parametric model results comparison



	Regression model	Neural network		
		Average	Max	Min
MAPE on training set	10%	2%	4%	1%
MAPE on test set	10%	7%	9%	7%
Generalization factor	50%	65%	80%	50%

Comparison of Results

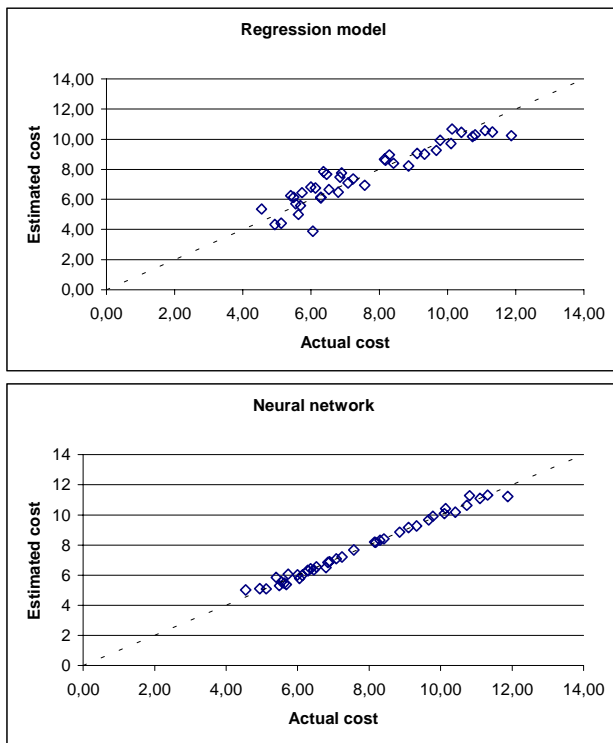
After the validation of the two models, the out-coming results have been compared considering the MAPE and the *Generalisation factor (Gf)*, defined as:

$$Gf = \frac{k}{M} \cdot 100 \tag{5}$$

where M is the number of patterns that compose the test set and k is the number of such patterns estimated with an error less then 10% (this value having been fixed as a threshold level).

The statistical analysis shows the superiority of the ANN model compared to the linear regression technique; the average MAPE on the training set falls from about 10% to about 2%. Figure 7 highlights the ANN model’s outperformance of the parametric model on the major part of the training set. The ANN model shows a better behaviour on the test set as well.

Figure 8. Actual cost vs. estimated cost



The excellent results on all the test samples (about 7.5%) of the experimental trials show the robustness of the ANN.

The analysis of the *Gf* shows that the performance of the ANN is better than that of the linear-regression model even on an unknown sample. The better performance indicates a better ability to generalise.

Finally, Figure 8 shows the actual cost versus the estimated cost with the two models; the closer the points to the dotted line, the better the estimation.

Conclusion

The adoption of cost-estimation predictive models in the first stages of the product development process is extremely important in order to provide a comprehensive technical and economical view of a new product concept according to a firm's overall

competitive strategy and key success factors. Knowing in advance the cause-and-effect relationship between design solutions and production costs is extremely useful both for internal manufacturing activities and for purchased parts.

The choice of the predictive model is generally based on the classical cost/benefit ratio; in this sense, regression models usually provide better results. However, more recently developed ANNs seem to represent a valid alternative, especially when the CER form is unknown and cannot be logically argued.

In the case study illustrated in this chapter, the ANN has shown better results in all the validation samples than a parametric model, without any significant variance problems (i.e., the dependence of the model on the data set used to construct it).

It is also interesting to extend the analysis beyond the quantitative data to include some qualitative considerations.

The most relevant point concerns the inherent logic of the two approaches. Whereas the use of a parametric model requires the specification of the analytical expression of the relationship that links input and output, the specification is not necessary with a neural network. In fact, ANNs are able to determine autonomously the most appropriate form of the relationship.

Summing up:

- The *ex ante* analysis of the problem is much leaner and faster and the outcome of very complex or innovative problems is independent from the ability of the analysts to find the key independent variables and the most appropriate kind of analytical expression.
- At the same time, a limit of the neural network approach is the impossibility to know the kind of relationship, since it is not clear how the results are achieved. In other words, in the neural network approach the object of analysis is treated as a “blackbox”; hence, it is impossible to give a theoretical interpretation of the obtained results, especially if there are unpredicted or (at least intuitively) unjustified values. This fact has often led to some scepticism about this methodology in several application contexts. The treatment is also due to the difficulty that the users of the methodology face when they are asked to prove the quality of the outcome in case of counterintuitive or questionable results.

Moreover, it could be objected that if the knowledge of the form of the relationship is not necessary to implement a neural network approach, it is nevertheless necessary to predetermine the structure of the network. The possible answers to this critical consideration are the following:

- The application contexts of the network structures that have been developed so far (e.g., multilayer, Adaptive Resonance Theory or ART, self-organising, etc.) are quite well-known, and the identification of the most appropriate structure is relatively simple.

- Software packages for the design of neural networks are generally coupled with tools aimed at evaluating the “learning attitude” of the network, and, at implementing the appropriate modifications if the obtained response is unsatisfactory.

The users of parametric models often cite the excellent (or at least satisfactory) quality/cost ratio. However, the implementation cost of a neural network is generally quite similar to that of a parametric model. Indeed, the lower cost of preliminary analysis is balanced by the higher costs of developing and testing the ANN. However, the higher robustness of the ANN methodology and the consequent higher propensity to deal with redundant or wrong information enables the elimination or consistent reduction of very time consuming and quite expensive data analysis.

Another strength of neural networks is their flexibility to changes made in the structure of the analysed system after the completion of the model’s development. For example, if the production process of a firm is modified through the implementation of new technologies, the parametric model must be completely revised and retested; but, by using a neural network, it will be sufficient to conduct a new training program with a new set of data (the structure of the network may not even be modified).

Finally, neural networks are completely data-driven and require the definition of a proper data set. Although data are also required in the CER-developing process (for example, for the estimation of the parameters of the model), the ANN application has to face the problem of overfitting, which could dramatically reduce the generalisation ability of the ANN. Using proper techniques, such as early stopping and validation data set, could smooth such an effect.

References

- Ansari, S. L., Bell, J. A., & the CAM-I Target Cost Core Group (Eds.). (1997). *Target cost: The next frontier in strategic cost management*. New York: Irwin.
- Berliner, C., & Brimson, J. A. (Eds.). (1988). *Cost management for today’s advanced manufacturing*. Boston: Harvard Business School Press.
- Blanchard, B. S. (1979). *Design and manage to life cycle cost*. Portland, OR: M/A Press.
- Cavalieri, S., Garetti, M., & Taisch, M. (1995, August 6-10). A predictive neural network modelling system for process plants. In *Proceedings of The 13th International Conference on Production Research, 1*, 571-573.
- Cavalieri, S., Rabe, M., & Taisch, M. (1997, March 18-21). A neural network approach to the plant design problem. In *Proceedings of The 1st International Conference on Engineering Design and Automation*, Bangkok, Thailand.
- Cavalieri, S., & Taisch, M. (1996, November 4-6). Neural networks in hybrid intelligent manufacturing systems. In *Proceedings of the APMS '96 — International Conference on Advances in Production Systems*, Kyoto, Japan.

- Chester, J. (1993). *Neural networks: A tutorial*. New York: Prentice Hall.
- Cooper, R. (1997). *Target costing and value engineering*. New York: Productivity Press.
- Fabrycky, W. J. (1991). *Life cycle cost and economic analysis*. New York: Prentice Hall.
- Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4, 1-58.
- Hammer, M., & Stanton, S. (1999, November-December). How process enterprise really work. *Harvard Business Review*, 108-118.
- Hill, T., O'Connor, M., & Remus, W. (1996). Neural network model for time series forecasts. *Management Science*, 42(7), 1082-1092.
- Hiramoto, T. (1988, July-August). Another hidden edge — Japanese management accounting. *Harvard Business Review*, 22-26.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359-366.
- Huang, G. Q. (1996). *Design for X: Concurrent engineering imperatives*. London: Chapman & Hall.
- Lawrence, S., Giles, C. L., & Tsoi, A. C. (1996). *What size neural network gives optimal generalization? Convergence properties of backpropagation*. (Technical Report UMIACSTR-96-22 and CSTR-3617). College, Park, MD: Institute for Advanced Computer Studies, University of Maryland.
- Mason, A. K., & Smith, A. E. (1997). Cost estimation predictive modeling: Regression versus neural network. *The Engineering Economist*, 42(2), 137-162.
- NASA. (1996). *Parametric cost estimating handbook*. Retrieved from <http://www.jsc.nasa.gov/bu2/pcehg.html>
- O'Rourke B. (1998). Neural nets forecast futures prices. *Financial Engineering News*. Retrieved from <http://fenews.com/1998/Issue3/029802.htm>
- Porter, M. E. (1980). *Competitive strategy—Techniques for analysing industries and competitors*. New York: The Free Press.
- Sakurai, M. (1989, Summer). Target costing and how to use it. *Journal of Cost Management*, 39-50.
- Shields, M. D., & Young, S. M. (1991, Fall). Managing product life cycle costs: An organisational model. *Journal of Cost Management*, 5(3), 39-52.
- Shtub, A., & Zimmerman, Y. (1993). A neural-network-based approach for estimating the cost of assembly systems. *International Journal of Production Economics*, 32(2), 189-207.
- Twomey, J. M., & Smith, A. E. (1993). Nonparametric error estimation methods for validating artificial neural networks. *Intelligent Engineering Systems Through Artificial Neural Networks*, 3, 233-238.
- Ulrich, K. T., & Eppinger, S. D. (1995). *Product design and development*. New York: McGraw-Hill International Editions.
- Zeleny, M. (1988). What is integrated process management. *Human Systems Management*, 7, 265-267

Zhang, Y. F., Fuh, J. Y., & Chan, W. T. (1996). Feature-based cost estimation for packaging products using neural networks. *Computers in Industry*, 32, 95-113.

Endnote

- ¹ It must be noticed that sometimes the term “redesign to cost” is used with a totally different meaning: it is referred to the redesign of business processes, and not of products, and includes all the organisational tools and rules aimed at the redesign of business processes in a cost-reduction perspective (someway similar to the BPR theory).

Chapter XIII

A Neural-Network-Assisted Optimization Framework and Its Use for Optimum-Parameter Identification

Tapabrata Ray, University of New South Wales, Australia

Abstract

Surrogate-assisted optimization frameworks are of great use in solving practical computationally expensive process-design-optimization problems. In this chapter, a framework for design optimization is introduced that makes use of neural-network-based surrogates in lieu of actual analysis to arrive at optimum process parameters. The performance of the algorithm is studied using a number of mathematical benchmarks to instill confidence on its performance before reporting the results of a springback minimization problem. The results clearly indicate that the framework is able to report optimum designs with a substantially low computational cost while maintaining an acceptable level of accuracy.

Introduction

There are numerous problems in the area of process design in which a designer is faced with the challenge to identify optimum process parameters that maximize one or more performance measures while satisfying constraints posed by statutory requirements, physical laws, and resource limitations. Currently, a vast majority of such applications are guided by trial and error and user experience. Such problems are nontrivial to solve as there are a large number of parameters that could be varied; the performance function is highly nonlinear and computationally expensive as it often involves calculations based on finite element methods (FEM), computational fluid dynamics (CFD), and so on.

Population-based, stochastic optimization methods like Genetic Algorithm (GA), Evolutionary Algorithm (EA), Differential Evolution (DE), and Particle Swarm Optimization (PSO) methods have been quite successful in solving highly nonlinear, mixed-variable optimization problems. However, all the aforementioned methods are known to be computationally expensive, as they need to sample numerous candidate solutions and hence cannot be used outright to deal with optimum process-parameter-identification problems involving computationally expensive simulations. In order to contain the computational time within affordable limits, two schemes are usually adopted within a population based stochastic algorithm, namely (a) use of multiple processors to evaluate different candidate solutions and (b) use of approximations (surrogate models) in lieu of actual expensive simulations.

In order to use approximations and surrogate models within an optimization framework, one needs to decide on the following: (a) representation accuracy of the surrogate model and (b) choice of a particular surrogate model. Surrogate models often have large approximation errors and can introduce false optima (Jin, Olhofer, & Sendhoff, 2002). Introduction of these false optima is a particularly serious problem when used in conjunction with stochastic optimization methods like GAs and EAs as they could converge incorrectly, referred to as ill-validation (Jin, Olhofer, & Sendhoff, 2000). The problem of ill-validation is seldom addressed in the literature, and most reported applications using approximate functions tend to use a once-for-all approximation function throughout the course of optimization without even a check on the validity of approximation at different stages of optimization (Jin et al., 2000). A naïve application of the approximate model repeatedly without retraining may thus lead to incongruity between the original and surrogate search spaces. Ratle (1998) suggested a heuristic convergence criterion used to determine the retraining frequency based on the convergence stability and the correlation between the actual and approximate function spaces.

The second issue relates to the choice of a surrogate model. The choice could range from Quadratic Response Surfaces, artificial-neural-network- (ANN-) based approximators like Multilayer Perceptrons (MLPs), Radial Basis Function Networks (RBFs), or geostatistical methods like Kriging and Cokriging. ANN-based approximators, that is MLPs and RBFs are particularly well suited for the present purpose as they are able to capture nonlinear relationships and known to be universal function approximators (Hornik, Stinchcombe, & White, 1989; Poggio & Girosi, 1989). An extensive discussion of these two networks can be found in Haykin (1999).

The following section presents the details of the neural-network-assisted framework. The optimization algorithm is a population-based, stochastic, zero-order, elite-preserving algorithm that makes use of approximate function evaluations in lieu of actual function evaluations. Two surrogate models are available for a designer to choose from: (a) RBF and (b) MLP. The surrogate model is periodically retrained after a few generations and a scheme based on controlled elitism is incorporated to ensure convergence in the actual function space. The performance of the surrogate assisted optimization framework is validated using mathematical test functions prior to its use in solving the springback minimization problem.

Numerical simulation of a sheet-metal-forming-process is important, as actual experimentation is expensive. Analytical solutions are limited due to the nonlinearity of the deformation process and the complexity of the shape of the dies. In a sheet-metal-forming operation, the blank material is closely formed to correspond to the die shape. However, when the load is released and the part is taken out of the press, there is often an unwanted change in shape. This phenomenon is known as springback, and it is a major quality concern in stamping operations. If the shape deviation due to springback exceeds the given tolerance, it could create serious problems for subsequent assembly operations. Therefore, the springback problem is of crucial practical importance. To compensate the shape deviation caused by springback, a common method is to modify the die topology. This is a challenging task and is largely made by the experienced designers using the trial-and-error method. Research on factors influencing springback has focused mainly on the geometric and material parameters, which are related to the tools and the blank sheet. In dealing with the springback problem, three approaches have been commonly used: analytical methods, experimental methods, and numerical methods. The pioneering work on the optimal process design for metal forming is led by Richmond and Devenpeck (1962), who applied the slip-line method to optimize forming energy. Roy, Ghosh, and Shivpuri (1996) employed the genetic algorithm for the design optimization of drawing and forging processes. In this study, a springback-minimization problem is solved using the surrogate assisted optimization framework and optimum process parameters are identified. It is clear that the framework is able to identify solutions with far less computational cost without significant deterioration in the solution quality. The following sections present the details of the springback minimization problem and list the major observations of this study.

Mathematical Model

A constrained, single-objective optimization problem in the context of minimization is presented next.

Minimize:

$$f(\mathbf{x}) \tag{1}$$

where $\mathbf{x} = [x_1, x_2, \dots, x_n]$ is the vector of n process parameters, $f(\mathbf{x})$ is the objective that needs to be minimized.

Subject to:

$$g_i(\mathbf{x}) \geq a_i, \quad i = 1, 2, \dots, q \tag{2}$$

where q is the number of inequality constraints.

For a set of M candidate solutions in a population, the objective can be represented as follows:

$$OBJECTIVE = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_M \end{bmatrix} \tag{3}$$

For each candidate solution, the constraint satisfaction vector $\mathbf{c} = [c_1, c_2, \dots, c_q]$ is given by:

$$c_i = \begin{cases} 0 & : \text{if constraint is satisfied} \\ a_i - g_i(\mathbf{x}) & : \text{if constraint is violated} \end{cases} \tag{4}$$

For the above c_i 's, $c_i = 0$ indicates the i^{th} constraint is satisfied, whereas $c_i > 0$ indicates the violation of the constraint. The constraint matrix for a population of M candidate solutions assumes the form:

$$CONSTRAINT = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{M1} & c_{M2} & & c_{Mq} \end{bmatrix} \tag{5}$$

Given a set of M candidate solutions, the feasible and the infeasible solutions are separated into two sets F and IF respectively. Let us assume that there are $M1$ solutions in Set F and $M2$ solutions in Set IF. The set of feasible solutions (F) are assigned ranks based on nondominance using (Nondominated Sorting) such that the best solution has a rank = 1 and the worst solution has a rank of R. The rank of the infeasible solutions is derived using nondominated sorting based on the constraint matrix. The rank of each infeasible solution is then incremented by a constant value (in this case it is R) that is equal to the rank of the worst feasible solution. The rank of every solution in the population is then converted to fitness as follows:

$$Fitness(i) = Max.Rank - 1 - Rank(i) \quad (6)$$

where *Max.Rank* is the Maximum Rank of an individual in the population. Ray, Tai, and Seow (2001) introduced the concept of handling constraints via nondominance.

The pseudocode of the algorithm is presented next and all other details of the mechanisms are described in subsequent sections.

START

Initialize a Population, $gen = 0$ and Specify λ, γ

Evaluate Individuals to compute $f_1(\mathbf{x})$ and constraints $\mathbf{c}(\mathbf{x}) = [c_1 \ c_2 \ \dots \ c_q]$

Create a Surrogate Model to Approximate $f_1(\mathbf{x})$ and $\mathbf{c}(\mathbf{x}) = [c_1 \ c_2 \ \dots \ c_q]$

While $gen < \lambda$ Do

$gen = gen + 1$

Rank Solutions

Preserve Elites

To Fill the remaining members of the Population

Do Select Parents for Mating via Roulette Wheel based on fitness

Generate a Child via Recombination

Call Surrogate Model to compute Performance of the Child.

End

If ($gen \bmod \gamma = 0$) Retrain the Surrogate Model

End While

END.

Where λ denotes the maximum number of generations allowed for the evolution process and γ denotes the periodic retraining frequency of the surrogate model.

Initialization

The solutions are created using Equation 7:

$$\tilde{x} = x_{low} + \delta(x_{upp} - x_{low}) \quad (7)$$

where \tilde{x} denotes the initialized variable, x_{low} and x_{upp} denotes the lower and upper bounds of the variable and δ is a uniform random number lying between 0 and 1. For practical problems, one could use sampling based on Design of Experiments (DOE) instead of random sampling.

Recombination

The recombination operator used in this study creates a child as follows:

1. Scale every variable between 0 and 1 using the maximum and minimum value of variables in the population.
2. $\mathbf{D} = \sum_{j=1}^M (I_L^j - I_F^j)^2$; $j = 1, \dots, M$ variables; I_L^j denotes the j^{th} variable of the leader (P) and I_F^j denotes the j^{th} variable of the follower (F).
3. $C(i) = P(i) + N(\mu = 0, \sigma) \cdot D$; where $s = 1.0$ is the variance of the normal distribution, $i = 1, \dots, M$ variables.
4. Transform $C(i)$ s back to original scale to get the new location of the individual F.

The user is free to choose or use any other recombination schemes like Parent Centric Crossover (PCX) or Simulated Binary Crossover (SBX).

K-Means Clustering Algorithm

The k-means clustering algorithm is used to identify k data sets that are used to create the surrogate model. Consider m data sets $\{x_1, x_2, \dots, x_m\}$ in n -dimensional space. We would like to obtain k centers, that is, $C = \{c_1, \dots, c_k\}$ using the k -means algorithm. The steps involved can be summarized as follows:

1. Assign first k datasets as k centers, i.e., $C = \{x_1, \dots, x_k\} = \{c_1, \dots, c_k\}$.
2. For each data point x_i , compute its membership function, ψ and weight, w :

$$\psi(\mathbf{c}_l | \mathbf{x}_i) = \begin{cases} 1; & \text{if } l = \arg \min_j \|\mathbf{x}_i - \mathbf{c}_j\|^2 \\ 0; & \text{otherwise} \end{cases} \quad (8)$$

$$w(\mathbf{x}_i) = 1 \quad \forall i = 1, \dots, m. \quad (9)$$

It can be seen from the definition of the membership function that k-means uses a hard membership and a constant weight function that gives all data points equal importance.

3. For each center \mathbf{c}_j , recompute its location from all data points \mathbf{x}_i according to their memberships and weights:

$$\mathbf{c}_j = \frac{\sum_{i=1}^m \psi(\mathbf{c}_j | \mathbf{x}_i) w(\mathbf{x}_i) \mathbf{x}_i}{\sum_{i=1}^m \psi(\mathbf{c}_j | \mathbf{x}_i) w(\mathbf{x}_i)} \quad (10)$$

4. Repeat steps 2 and 3 until convergence is achieved. Usually this is done by ensuring the membership function is unchanged for all data points between iteration. The k-means clustering is a popular choice as it is easy to understand and implement.

Radial Basis Function Network

Radial basis functions belong to the class of artificial neural networks and are a popular choice for approximating nonlinear functions. In this section, the necessary details of implementing a radial basis function network are included for completeness. A radial basis function (RBF) ϕ is one, whose output is symmetric around an associated center, $\boldsymbol{\mu}$. That is: $\phi(\mathbf{x}) = \phi(\|\mathbf{x} - \boldsymbol{\mu}\|)$, where the argument of ϕ is a vector norm. A Gaussian function has been used for the RBF by selecting $\phi(r) = e^{-r^2/\sigma^2}$, where σ is the width or scale parameter. A set of RBFs can serve as a basis for representing a wide class of functions that are expressible as linear combinations of the chosen RBFs:

$$y(\mathbf{x}) = \sum_{j=1}^m w_j \phi(\|\mathbf{x} - \mathbf{x}_j\|) \quad (11)$$

However Equation 11 is usually very expensive to implement if the number of data set is large. Thus a generalized RBF network is usually adopted of the following form:

$$y(\mathbf{x}) = \sum_{j=1}^k w_j \phi(\|\mathbf{x} - \boldsymbol{\mu}_j\|). \quad (12)$$

Here k is typically smaller than m and w_j , which are the unknown parameters that are to be “learned.” The k number of dataset is determined from the k -means clustering mentioned previously. The training is usually achieved via the least square solution:

$$\mathbf{w} = \mathbf{A}^+ \mathbf{d}. \quad (13)$$

Here \mathbf{A}^+ is the pseudoinverse and \mathbf{d} the target output vector. The pseudoinverse is used as typically \mathbf{A} is a rectangular matrix and thus no inverse exists. However, the computation of the pseudoinverse requires a matrix inversion that is computationally expensive for large problems and thus the recursive least-squares estimation has been often used.

Multilayer Perceptron

Multilayer perceptrons are also quite popularly used as generic function approximators. The number of input nodes of a MLP is equal to the number of independent variables, while the number of outputs is equal to the number of functions being approximated by the network. The layers lying between the input and the output layers are referred to as hidden layers and the complexity of a network depends on the number of such layers and the number of nodes in each of them. The predictive capability of the network is captured by the nodal interconnections (weights) and the transfer functions. The process of neural network training refers to identifying the best set of weights such that the error in prediction is minimum. Ray, Gokarn, and Sha (1996) proposed the use of Modified Marquardt Levenberg algorithm for a faster training of neural networks. The capability of the network for prediction at the end of its learning process is tested on a test data set that is exclusive from the training data set.

Table 1. List of test functions

Test Function	Expression
Spherical	$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$
Ellipsoidal	$f(\mathbf{x}) = \sum_{i=1}^n ix_i^2$
Schwefel	$f(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$
Rosenbrock	$f(\mathbf{x}) = \sum_{i=1}^{n-1} (x_i - 1)^2 + 100(x_i^2 - x_{i+1})^2$
Rastrigin	$f(\mathbf{x}) = 10n + \sum_{i=1}^n (x_i^2 - 10 \cos(2\pi x_i))$

Numerical Examples

In the following section, the behavior of the surrogate assisted optimization model is studied using five 20-dimensional mathematical test functions and subsequently the springback-minimization problem is solved to demonstrate the efficacy of the proposed method.

A detailed study on the performance of the surrogate assisted optimization framework appears in Won, Ray, and Kang (2003).

Mathematical Test Functions

The test functions are Spherical, Rosenbrock, Rastrigin, Schwefel, and Ellipsoidal, and their forms are listed in Table 1. All numerical simulations are carried out using the optimization framework with the following assumptions:

- The population size of $10n$ was used for all the simulation.
- s , where n is the number of variables for the problem.
- For all the test cases, 10 independent runs were conducted with the number of generations being 1,000.
- The search space was between $[-5.12, 5.12]$ for the Spherical, Ellipsoidal, Schwefel, and Rastrigin functions while a search space of $[-2.048, 2.048]$ was selected for the Rosenbrock function in accordance to the convention in the literature.
- A radial-basis function (RBF) network was used with $5n$ centers and two nearest neighbor in accordance with Haykin (1999).
- The parent-centric crossover (PCX) operator was used to create a child from three parents.
- Retraining the RBF network was done after every 10 generations.
- The entire simulation process was executed using a Pentium® 4, 2.4GHz CPU processor.

Results presented in Table 2 indicate the performance of the optimization algorithm (OA) when actual evaluations have been used throughout the course of optimization. The number of actual function evaluations used by the OA model is listed in Column 3 of Table 3. To achieve the same mean level of convergence as compared to the OA model, the RBF-OA model typically uses around 50% less actual function evaluations as shown in Column 4 of Table 3. The number of approximations used by the RBF-OA model is listed in Column 5 of Table 3. The computational time required by the OA model and the RBF-OA model is presented in Table 4 and Table 5. The results of the surrogate-assisted optimization framework on these 20-dimensional highly nonlinear problems clearly show

Table 2. Statistics for 20-D results using Optimization Algorithm (OA)

Test Function	Best Fitness	Worst Fitness	Mean Fitness	Median Fitness	Standard Deviation
Spherical	3.3851×10^{-22}	1.0224×10^{-20}	2.9952×10^{-21}	1.9470×10^{-21}	2.7881×10^{-21}
Ellipsoidal	1.5937×10^{-10}	3.7958×10^{-7}	4.7247×10^{-8}	9.3724×10^{-9}	1.1121×10^{-7}
Schwefel	1.3206×10^{-6}	5.9133×10^{-5}	2.0129×10^{-5}	1.0319×10^{-5}	1.987×10^{-5}
osenbrock	14.9216	19.5135	17.6649	17.5303	1.2013
Rastrigin	10.0065	39.2395	22.0698	23.3051	7.8031

Table 3. Function evaluations required for same tolerance (20-D)

Test Function	Tolerance	Optimization Algorithm (Actual function evaluations)	Surrogate Assisted (Actual function evaluations)	Surrogate Assisted (Approx. function evaluations)
Spherical	2.9952×10^{-21}	199200	110467	1091640
Ellipsoidal	4.7247×10^{-8}	199200	81534	805200
Schwefel	2.0129×10^{-5}	199200	144267	1426260
Rosenbrock	17.6649	70447	21201	207900
Rastrigin	22.0698	101650	28020	213040

Table 4. Summary of 20-D computational efforts using Actual Evaluations (OA)

Number of actual function evaluations	199200
Total time for Actual Evaluations	37.311s
Total elapsed time (Wall clock time)	420.315s

Table 5. Summary of 20-D computational efforts using approximations (RBF-OA)

Number of actual function evaluations	20201
Number of approximate function evaluations	198000
Total time for training with RBF	77.363s
Total time for RBF approximations	416.851s
Total time for actual function evaluations	3.643s
Total elapsed time (Wall clock time)	887.537s

Table 6. Material properties of the sheets

	Aluminum (1)	AK Steel (2)	HT Steel (3)
Young's Modulus (GPa)	69	206	206
Poisson's Ratio	0.330	0.300	0.300
Strength coefficient (MPa)	570	567	671
Hardening exponent for yield strength	0.347	0.264	0.219
Flow potential exponent in Barlat's model	8	6	6
Anisotropy Coefficient	0.710	1.770	1.730
Sheet Thickness (mm)	1.00	1.00	1.00

that a saving of nearly 50% of actual function evaluations is possible while maintaining an acceptable accuracy. This is of great significance as it could mean cutting down expensive CFD or FEM computations while maintaining an acceptable accuracy.

Springback Minimization

In this section, the springback-minimization problem is modeled and solved using an MLP-assisted optimization algorithm although one could use an RBF-assisted model, too. This problem has been modeled as a seven-variable, unconstrained discrete optimization problem where the first variable is the material type, while the other six are process parameters. LS-DYNA3D has been used to model springback and a simple axisymmetric stamping problem was selected for numerical study. In this study, a blank sheet with dimension 177.8 × 177.8mm, punch speed of 25.4mm/min; and a range of punch penetrations between 1.27 to 21.6 mm was used. Springback is calculated with an implicit analysis, which can simulate the unloading of the blank, due to the removal of the tools. The accuracy of the prediction of springback by numerical simulation depends strictly on the accuracy of the stress distribution computed by deep drawing simulation, which is very sensitive to the accuracy with which the process is modeled. The methodology employed for the validation of the simulation codes is based on the definition of the numerical parameters suited to reduce CPU time, without affecting the accuracy of the simulation results. In this approach, the dome height was adopted as the metric for evaluation of springback.

The problem of springback minimization is solved using the following methods: Random Search (RS), Optimization Algorithm (OA) with all actual computations, and Surrogate Assisted Optimization Algorithm (MLP-OA). The material variables and process parameters with their possible states are listed in Tables 6 and 7, respectively. For this single-

Table 7. Set of process parameters

	P1	P2	P3	P4	P5	P6
1	0.10	0.15	0.15	0.20	0.10	0.20
2	0.04	0.10	0.0	0.10	0.15	0.15
3	0.02	0.10	0.15	0.20	0.02	0.12
4	0.11	0.12	0.15	0.15	0.05	0.10
5	0.12	0.15	0.13	0.10	0.15	0.20
6	0.10	0.11	0.20	0.17	0.20	0.20
7	0.15	0.14	0.20	0.18	0.14	0.11
8	0.16	0.10	0.15	0.14	0.14	0.15
9	0.15	0.20	0.12	0.11	0.10	0.10
10	0.10	0.18	0.11	0.10	0.12	0.15
11	0.15	0.10	0.09	0.14	0.16	0.11
12	0.13	0.16	0.10	0.12	0.15	0.13
13	0.14	0.12	0.18	0.15	0.10	0.11
14	0.15	0.15	0.20	0.20	0.15	0.15
15	0.08	0.10	0.10	0.11	0.16	0.15

P1: Dynamic Friction on Punch/Sheet

P2: Static Friction on Punch/Sheet

P3: Dynamic Friction on Die/Sheet

P4: Static Friction on Die/Sheet

P5: Dynamic Friction on Holder/Sheet

P6: Static Friction on Holder/Sheet

objective, unconstrained, discrete springback-minimization problem there are 571,536 solutions. Table 8 presents the results obtained using various approaches.

Random Search: With a random search at 10 points, the minimum and maximum springback values are 0.3890mm and 0.9390mm. With a random search at 70 points, the best springback is 0.3830mm while the worst is 0.9440.

Optimization Algorithm: Figure 1 presents the springback values of the solutions in the initial population. It can be observed from Figure 1 that there are three distinct bands of

Table 8. Comparison of results

Method	Function Evaluations	Function Value (Springback) (mm)
Random Search	10	Min.: 0.3890 Max.: 0.9390
Random Search	25	Min.: 0.3870 Max.: 0.9390
Random Search	70	Min.: 0.3830 Max.: 0.9440
Optimization Algorithm	25	Min.: 0.3840 Max.: 0.3970
Optimization Algorithm	169	Min.: 0.3830 Max.: 0.3880

Table 9. Top three solutions from OA and MLP-OA

Methods	Springback (mm)
OA 1	0.3830 (Actual)
OA 2	0.3840 (Actual)
OA 3	0.3850 (Actual)
MLP-OA 1	0.3817/0.3870 (Predicted/Actual)
MLP-OA 2	0.3818/0.3850 (Predicted/Actual)
MLP-OA 3	0.3823/0.3880 (Predicted/Actual)

Figure 1. Springback (initial population)

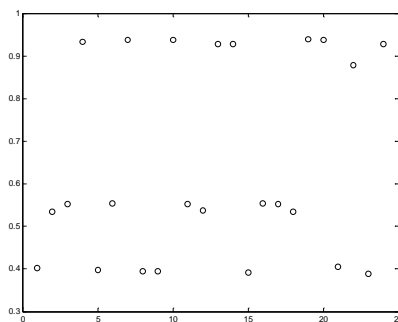
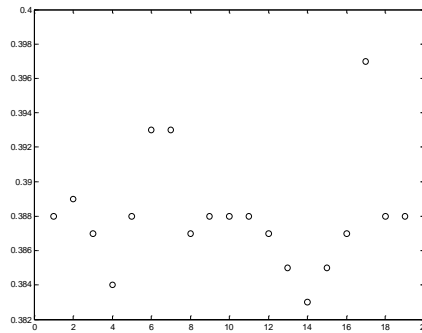


Figure 2. Springback (final population)



springback values varying between 0.38mm and 1.0mm. With a mere 25 function evaluations, OA reports a solution with springback of 0.3840 mm, which is 0.26% worse than the best value of 0.3830mm. The maximum springback value of the solutions in the final population is 0.3970mm, which indicates a good convergence of the solutions to the lowermost band. On increasing the number of function evaluations to 169, the best solution has a springback of 0.3830mm while the maximum springback in the final population is 0.3880mm. The top three solutions are listed in Table 9. The average springback computation takes around 12 minutes of CPU time. The springback values of the solutions of the final population are presented in Figure 2. It can be noted that the springback values of the solutions of the final population lie between 0.3830 and 0.3970, clearly indicating that the solutions improved over the generations and reached the lowermost band of springback.

Multilayer Perceptron Embedded Optimization Algorithm: The MLP of the MLP-OA model has been trained using 70 data sets, while its prediction capabilities are tested on 25 data sets that are exclusive of the training set. Two architectures are tested—one having five hidden-layer neurons, while the other had four hidden-layer neurons. The prediction capabilities of the network are illustrated in Figures 3 and 5 on training sets and Figures 4 and 6 on test-data sets for the above architectures. The CPU time for training the 7x4x1 and the 7 x5x1 neural network architectures are 61.193 seconds and 118.616 seconds respectively on SGI Origin 2000. It can be observed from Figures 4 and 6 that the neural networks are capable of predicting reasonable accurate springback values (within +/- 2%) that can be used by the optimization algorithm as an approximation. In order to study the behavior of the MLP-OA model, the springback-evaluation process is removed and the springback-estimation process is introduced using the 7x5x1-network architecture. Using a population of 25 individuals and with 67 calls to the approximator, the algorithm converged to the final population. The plot of the initial and final population using the MLP-OA model is shown in Figures 10 and 11. It can be observed that MLP-OA is capable of minimizing the springback and converge to a final population just like the optimization algorithm (OA) alone. The introduction of the neural-network-based approximator within the optimization algorithm resulted in the reduction of actual

springback computations from 169 to 70 (as the network was trained on 70 data sets) that translates to about 1,200 minutes of CPU time saving. Table 4 presents the top three alternatives as obtained by the optimization algorithm (OA) and the MLP-OA.

Discussion and Conclusion

Optimum product and process-design problems are known to involve computationally expensive analysis and exhibit a highly nonlinear functional behavior. In order to effectively and efficiently solve such problems, surrogate assistance plays an important role. It is clearly visible from the studied examples that the surrogate-assisted optimization framework could arrive at competitive solutions with far less number of actual function calls. It is also interesting to observe that both the neural-network-based approximators (RBF and MLP) could approximate nonlinear functions reasonably well for both the mathematical test functions and the springback-minimization problem. The process of periodic retraining of the MLPs and RBFs within the framework ensures adequate generalization of the network at different stages of the optimization process and is the key to avoid ill-validation problems.

Although a single objective, unconstrained, discrete problem of springback minimization was solved, constrained optimization problems and problems with multiple objectives could be solved using the same framework. It can be seen that with the introduction of the neural-network-based approximator, there is a significant reduction in the number of actual springback computations and thus the CPU time needed to optimize the design. Although the algorithm can be easily ported to run on multiple processors, the speedup is limited by the number of commercial licenses of the analysis codes that an organization might have. We are currently working on the optimization framework via approximations to deal with shape optimization problems for aerospace applications.

References

- Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall Internal Inc.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359-366.
- Jin, Y., Olhofer, M., & Sendhoff, B. (2000). On evolutionary optimization with approximate fitness functions. In *Proceedings of Genetic and Evolutionary Computation Conference* (pp. 786-792). Morgan Kaufmann.
- Jin, Y., Olhofer, M., & Sendhoff, B. (2002). A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5), 481-494.

- Liew, K. M., Ray, T., Tan, H., & Tan, M. J. (2002). Evolutionary optimization and use of neural network for optimum stamping process design for minimum springback. *Transaction ASME Journal of Computing and Information Science in Engineering*, 2, 38-44.
- Poggio, T., & Girosi, F. (1989). Networks and the best approximation property. *Biological Cybernetics*, 63, 169-176.
- Ratle, A. (1998). Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In A. Eiben, Th. Bäck, M. Schoenauer, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature* (pp. 87-96). Berlin: Springer.
- Ray, T., Gokarn, R. P., & Sha, O. P. (1996). Neural network applications in naval architecture and marine engineering. *Artificial Intelligence in Engineering*, 1, 213-226.
- Ray, T., Tai, K., & Seow, K. C. (2001). Multiobjective design optimization by an evolutionary algorithm. *Engineering Optimization*, 33, 399-424.
- Richmond, O., & Davenpeck, M. L. (1962). A die profile for maximum efficiency in strip drawing. In *Proceedings of the Fourth U.S. Congress. Applied Mechanics, ASME* (pp. 1053).
- Roy, S., Ghosh, S., & Shivpuri, R. (1996). Optimum design of process variables in multi-pass wire drawing by genetic algorithms. *Journal of Manufacturing Science and Engineering*, 118, 244-251.
- Won, K. S., Ray, T., & Kang, T. (2003, December 6-8). A framework for optimization using approximate functions. In *Proceedings of the IEEE Congress on Evolutionary Computing*, Canberra, Australia.

Chapter XIV

Artificial Neural Networks in Manufacturing: Scheduling

George A. Rovithakis, Aristotle University of Thessaloniki, Greece

Stelios E. Perrakis, Technical University of Crete, Greece

Manolis A. Christodoulou, Technical University of Crete, Greece

Abstract

In this chapter, a neuroadaptive scheduling methodology, approaching machine scheduling as a control-regulation problem, is presented and evaluated by comparing its performance with conventional schedulers. Initially, after a brief reference to the context of existing solutions, the evaluated controller is thoroughly described. Namely, the employed dynamic neural network model, the subsequently derived continuous time neural network controller and the control input discretization that yield the actual dispatching times are presented. Next, the algorithm guaranteeing system stability and controller-signal boundedness and robustness are evaluated on an existing industrial test case that constitutes a highly nonacyclic deterministic job shop with extremely heterogeneous part-processing times. The major simulation study, employing the idealistic deterministic job-shop abstraction, provides extensive

comparison with conventional schedulers, over a broad range of raw-material arrival rates, and through the extraction of several performance indices verifies its superb performance in terms of manufacturing-system stability and low makespan, low average lead times, WIP, inventory, and backlogging costs. Eventually, these extensive experiments highlight the practical value and the potential of the mathematical properties of the proposed neuroadaptive controller algorithm and its suitability for the control of nontrivial manufacturing cells.

Introduction

Production scheduling deals with the allocation of the available resources over time for the manufacture of goods. It involves the decision-making mechanism whose objective is finding a way to assign and use the sequence of shared resources (labor, material, equipment), such that production constraints are satisfied and production costs are minimized.

In this chapter we address a machine-scheduling problem that, while constituting a simplified formalism of the production scheduling, still captures its fundamental complexity. More precisely, we focus on the deterministic job-shop scheduling, whereas a set of n jobs is processed on a finite set of m machines, with precedence constraints imposed on the sequence of individual operations.

The examined scheduling problem, deterministic job-shop scheduling, is the most general classical scheduling problem and due to its factorial explosion is classified into the large class of intractable numerical problems (NP) known as NP-hard, that is, problems that cannot be solved in time polynomial to the dimension of the problem under consideration. Job-shop scheduling due to its importance to the efficient management of manufacturing processes has been addressed by a plethora of approaches.

Next, a reference to industrial practice and to the existing approaches to job-shop scheduling is made, and the essence of our proposed scheduler along with the intended evaluation methodology is outlined.

Background

Current industrial practice has been mainly based on assisting experienced human schedulers with major software packages that implement distinct scheduling philosophies like manufacturing resource planning (MRP), just-in-time (JIT) production (Schonberger, 1983), and optimized production timetables (OPT), while more recently enterprise-resource planning systems (ERPs) are utilized in process industries (James, 1996).

Although production scheduling has been traditionally addressed by management science, operations research, and industrial engineering, its complexity and importance

have recently concentrated the efforts of different research communities concerned with artificial intelligence (Kusiak, 1987), dynamic programming, queuing-network theory (Jackson, 1963), systems simulation, large-scale systems, control theory, and other branches of engineering and computer science (Gupta, Evans, & Gupta, 1991; Rodammer, 1988).

In this work, we specifically focus on the deterministic job-shop scheduling. Job-shop scheduling due to its importance has been addressed by a plethora of approaches. Some of the elder techniques have been enumerative algorithms that provide exact solutions either by means of elaborate and sophisticated mathematical constructs — such as linear programming (Lawler, Lenstra, Rinnooy, & Shmoys, 1993), decomposition techniques, and Lagrangian relaxation—or by means of the branch and bound enumerative strategy, which involves search of a dynamically constructed tree that represents the solution space (Brandimarte & Villa, 1995). Limitations of the aforementioned enumeration techniques has led to suboptimal approximation methods, such as priority dispatch rules, that involve assignment of priority to each job primarily via heuristics (Panwalkar & Iskander, 1977), while recently, approaches employing fuzzy-logic techniques have emerged (Grabot & Geneste, 1994). Scheduling has been dominated by a set of innovative heuristic-approximation algorithms including the shifting-bottleneck procedure (Adams, Balas, & Zawack, 1988), tabu search (Glover, 1989), simulated annealing (Van Laarhoven, 1988), and genetic algorithms (Cheng et al., 1999). Furthermore, artificial intelligence methods have been applied ranging from neural networks (NNs) (Kim, Lee, & Agnihotri 1995; Sabuncuoglu & Gurgun, 1996) to constraint satisfaction techniques and expert systems. Recently, hybrid techniques that involve searching strategies that navigate heuristic algorithms in a problem domain away from local optima have been applied. Such techniques are genetic local search (Yamada & Nakano, 1996), and large-step optimization (Lourenco, 1995).

In this chapter, we present and systematically evaluate a novel neuroadaptive scheduling methodology (Rovithakis, Gaganis, Perrakis, & Christodoulou, 1999) by considering its application on a challenging existing industrial test case (Rovithakis, Perrakis, & Christodoulou, 2001). The examined neural network scheduler approaches the production-scheduling problem from a control-theory viewpoint (Gershwin, Hildebrant, Suri, & Mitter, 1986), in which scheduling is considered a dynamic activity. Thus, by defining release and dispatching times, setup times and maintenance as control input, and levels of inventory and machine status as system states, scheduling can be considered either as a regulation or tracking problem, where the requirements are to drive the state vector to some desired value (production requirement) or to follow some distributed over-time trajectory (Ioannou & Sun, 1995), respectively.

By taking advantage of current experience in the neuroadaptive control field (Rovithakis & Christodoulou, 1994, 1995, 1997) based on dynamic neural networks (NN), the deterministic job-shop scheduling problem has been considered a control-regulation problem, where system states (buffer levels) have to reach some prespecified values by means of control input commands. Based on a dynamic neural network model of the buffer states, derived in Rovithakis, Perrakis, and Christodoulou (1996, 1997, 1999), an adaptive continuous-time neural network controller has been developed. Dispatching commands are issued by means of a discretization process of the continuous-control input, which is defined as the operating frequency with which each distinct manufacturing operation

must occur while the controller guarantees the uniform ultimate boundedness of the control error as well as the boundedness of all other signals in the closed loop.

Further evaluation of the neural network scheduler is pursued by applying it on real data derived from a challenging manufacturing system (Rovithakis et al., 2001). The selected test case — the mechanical workshop of a German company — constitutes a complex job shop with extremely heterogeneous part-processing times, with 18 product types that may visit 18 different machines having to be produced, thus demanding sequencing of a total of 273 jobs, that is the production of a total of 273 parts. The performance of the algorithm is compared with modified versions of the well-established conventional scheduling policies First In First Out (FIFO), Clear a Fraction (CAF), and Clear Largest Buffer (CLB). All schedulers are compared over a range of raw-material-arrival rates and their performance is evaluated by means of the observed makespan, work in process (WIP), inventory, backlogging costs, and average lead times.

Thus, the derived simulation results, revealing superb performance in issues of manufacturing-system stability, low WIP, average lead times, backlogging and inventory costs for the NN scheduler, establish the proposed scheduler's applicability on the control of nontrivial manufacturing cells.

The structure of the chapter proceeds as follows: In "Problem Formulation and the DNN Architecture," a description of the proposed NN scheduler is presented. In "Test Case: SHW Mechanical Workshop," the scheduling problem for the selected test case is defined and the conventional schedulers employed to facilitate comparisons in this study are described. In "Results," critical presentation of the derived results is provided, and we conclude in the final section.

Problem Formulation and the DNN Architecture

This section is devoted to a brief presentation of the evaluated approach on the scheduling problem in manufacturing cells. These results, regarding the application of dynamic NNs to adaptively control FMS have been recently introduced in Rovithakis, Perrakis, and Christodoulou (1996, 1997, 1999).

Problem Formulation

The considered manufacturing systems resemble job shops in their layout, consist of M machines, and produce a multiple of P part types. Each part type requires a number of operations to be performed in a given sequence, defined by its route. Multiple routing is possible and parts may visit some machines several times. Each machine m is assumed to consist of a number $N(m)$ of submachines equal to the number of different part types it is able to process. Each submachine actually represents an operating mode of machine m . Let $O(m)$ be a set such that $s \in O(m)$ implies that submachine s is a submachine of the

actual machine m . The cardinality of $O(m)$ is equal to $N(m)$. Let also $N_M = \sum_{i=1}^M N(m_i)$. Only one submachine is allowed to be working at a time, processing parts of a single type. Machine operation times are supposed to be constant, that is deterministic, where different operation times for every submachine are allowed. Set-up times are assumed to be insignificant. A machine will be called “idle with respect to a part type”, if it is idling or is processing some part type different from the specified one.

The objective is to steer, through appropriate scheduling (sequences of dispatching commands), the manufacturing-system output buffers to reach a small neighborhood of predefined finished products, while keeping all intermediate buffers within their acceptable values.

Continuous Control Input Definition

It is assumed that an equivalent machine-operation frequency is used as a control input to the system, defined as the inverse of the time between two successive machine-starts. Using this definition, frequencies range between zero and a fixed finite value. The lower bound equal to zero, corresponds to an infinite idling time. The upper bound u_{\max} , corresponds to a minimum (zero) idling time-thus a maximum working rate, and equals to the reciprocal of machine operation time. It should be observed that this definition is a crucial point in our work, since it enables the transition from a discrete-event system to a continuous time one.

The Manufacturing Cell Dynamic Model

Recalling from (Rovithakis, Perrakis, & Christodoulou, 1996, 1997, 1999), the dynamic model from the evolution of the level x_i of a buffer, is assumed to be given by:

$$\dot{\bar{x}}_i = f_{0_i}(\bar{x}_i, u_i)u_i + f_{1_i}(\bar{u}_i) \quad (1)$$

where \bar{x}_i is the vector containing the levels of all directly connected preceding buffers, \bar{u}_i is the vector containing the frequencies of all submachines collecting products from x_i , u_i is the frequency of the submachine feeding buffer i , and $f_{0_i}(\cdot)$ $f_{1_i}(\cdot)$ are unknown functions of their arguments, specifically $f_{0_i}(\cdot)$ is the increasing rate of buffer i , and $f_{1_i}(\cdot)$ is its decreasing rate.

Since both $f_{0_i}(\cdot)$ $f_{1_i}(\cdot)$ are unknown functions of their arguments, neural networks of the form described in the next subsection are employed to obtain an accurate model for the manufacturing cell.

The Dynamic Neural Network Architecture

The dynamic neural networks that are used, called *Recurrent High-Order Neural Networks (RHONN)*, are fully interconnected nets, containing dynamical elements in their neurons. Therefore, these neural networks are described by the following set of differential equations:

$$\dot{\mathbf{x}} = W_0 S_0(x, u) u + W_1 S_1(u) \quad (2)$$

where $\dot{\mathbf{x}} \in \mathfrak{R}^B$, the inputs $u \in \mathfrak{R}^B$, W_0 and W_1 are $B \times L$ and $B \times L_0$ matrices respectively of adjustable synaptic weights. $S_0(x, u)$ is a $L \times B$ matrix with elements $S_{im}(z)$, $i = 1, 2, \dots, L$, $m = 1, 2, \dots, B$, of the form:

$$S_{im}(z) = \prod_{j \in I_{im}} [s(z_j)]^{d_j(i,m)} \quad (3)$$

where I_{im} , $i = 1, 2, \dots, L$ and $m = 1, 2, \dots, B$ are collections of $L \times B$ not ordered subsets of $\{1, 2, \dots, B\}$, $d_j(i, m)$ are non-negative integers and $z = [x, u]$. Similarly, $S_1(u)$ is a L_0 -dimensional vector with elements $S_k(u)$ of the form:

$$S_k(u) = \prod_{j \in I_k} [s(u_j)]^{d_j(k)} \quad (4)$$

For all $k = 1, 2, \dots, B$ where I_k are collections of L_0 not-ordered subsets of $\{1, 2, \dots, B\}$ and $d_j(k)$ are nonnegative integers. In both (2.3) and (2.4) $s(z_j)$ is a monotone increasing, smooth, function, which is usually represented by sigmoidals of the form:

$$S(z_j) = \frac{\mu}{1 + e^{-l_0 z_j}} + \lambda \quad (5)$$

For all $j = 1, 2, \dots, 2B$, with the parameters μ , l_0 , to represent the bound and slope of the sigmoid's curvature and λ a bias constant. Equation (2.2) can also be written in the equivalent form:

$$\dot{\mathbf{x}}_i = W_{0i}^T S_{0i}(\bar{x}_i, u_i) u_i + W_{1i}^T S_{1i}(\bar{u}_i) \quad (6)$$

For all $i = 1, 2, B$. For the neural network model, there exists the following approximation theorem (Kosmatopoulos, Polycarpou, & Christodoulou, 1995):

Theorem 1: Suppose that the system (1) and the model (6) are initially at the same state $x_i(0) = \dot{\bar{x}}_i(0), \forall i = 1, 2, \dots, B$. Then for any $\varepsilon_i > 0, i = 1, 2, \dots, B$ and any finite $T > 0$, there exists integers L, L_0 and matrices W_{0i}^*, W_{1i}^* , such that the state $\dot{\bar{x}}_i(t)$ of the dynamic neural network model (6) with $L \times B + L_0$ higher order connections and weight values $W_0 = W_{0i}^*, W_1 = W_{1i}^*$, satisfies:

$$\sup_{0 \leq t \leq T} |\dot{\bar{x}}_i(t) - x_i(t)| \leq \varepsilon_i$$

Theorem 1 prerequisites that the unknown vector fields $f_{0i}(\bar{x}_i, u_i), f_{1i}(\bar{u}_i), i = 1, 2, \dots, B$ are continuous and satisfy a local Lipschitz condition such that (1) has a unique solution in the sense of *Caratheodory* (Hale, 1969) Moreover, the previous theorem proves that if sufficiently large number of higher order connections are allowed in the dynamic neural network model, then it is possible to approximate a very large class of dynamical systems of the form (1), to any degree of accuracy.

Due to the approximation capabilities of the dynamic neural networks, it can be assumed, with no loss of generality, that the unknown system (1) can be completely described by a dynamic neural network plus a modeling error term $\varepsilon_i(\bar{x}_i, \bar{u}_i, u_i)$. In other words, there exist weight values W_{0i}^* , and W_{1i}^* such that the system (1) can be written as:

$$\dot{\bar{x}} = W_{0i}^{*T} S_{0i}(\bar{x}_i, u_i) u_i + W_{1i}^{*T} S_{1i}(\bar{u}_i) + \varepsilon_i(\bar{x}_i, \bar{u}_i, u_i) \quad (7)$$

where \bar{x}_i is the vector containing the levels of all directly connected preceding buffers, \bar{u}_i is the vector containing the frequencies of all submachines collecting products from x_i , u_i is the frequency of the submachine feeding buffer i , and $\varepsilon_i(\cdot)$ is a modeling error term assumed bounded, i.e. $\|\varepsilon_i(\cdot)\| \leq \varepsilon_i$ with ε_i arbitrarily small. Finally W_{0i}^* and W_{1i}^* represent the optimal weight values (as optimal we define these weight values that lead to minimum modeling error ε_i) of the NN and S_{0i}, S_{1i} are vectors that contain sigmoid functions.

Continuous Time Control Law

Let by $x_{i(t)}$ $i = 1, 2, \dots, B$ with B the total number of buffers, denote the target value for each buffer. Define the control error e_{c_i} as $e_{c_i} \stackrel{\Delta}{=} x_i - x_{i(t)}$. To derive stable control and update laws Lyapunov stability theory is employed following the analysis presented in Rovithakis, Perrakis, & Christodoulou (1997, 1999), the control and weight update laws listed later are derived:

$$u_i = -q_i \operatorname{sgn}(W_{0i}^T S_{0i}(\bar{x}_i, u_i)) \operatorname{sgn}(e_{c_i}) \quad (8)$$

$$q_i = \frac{1}{w_i^-} \left[W_{0i}^T \|S_{0i}(\bar{u}_i)\| + \gamma |e_{c_i}| \right] \quad (9)$$

$$\operatorname{sgn}(W_{0i}^T S_{0i}(\bar{x}_i, u_i)) = \begin{cases} 1 & \text{if } W_{0i}^T S_{0i}(\bar{x}_i, u_i) \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (10)$$

$$\operatorname{sgn}(e_{c_i}) = \begin{cases} -1 & \text{if } e_{c_i} < 0 \\ 0 & \text{if } e_{c_i} = 0 \\ 1 & \text{if } e_{c_i} > 0 \end{cases} \quad (11)$$

where γ , w_i^- are design constants.

$$\bar{W}_{0i} = \begin{cases} e_{c_i} S_{0i}(\bar{x}_i, u_i) u_i & \text{if } W_{0i} \in W_{0i} \\ & \text{or } W_{0i} \in \partial W_{0i} \text{ and } e_{c_i} S_{0i}^T(\bar{x}_i, u_i) u_i W_{0i} \geq 0 \\ e_{c_i} S_{0i}(\bar{x}_i, u_i) u_i - e_{c_i} S_{0i}^T(\bar{x}_i, u_i) u_i W_{0i} \left(\frac{1 + \|W_{0i}\|}{w_{0i}} \right)^2 W_{0i} & \\ & \text{if } W_{0i} \in \partial W_{0i} \text{ and } e_{c_i} S_{0i}^T(\bar{x}_i, u_i) u_i W_{0i} < 0 \end{cases} \quad (12)$$

$$\vec{W}_{li} = \begin{cases} D_1 & \text{if } W_{li} \in W_{li} \\ & \text{or } W_{li} \in \partial W_{li} \text{ and } C_r \geq 0 \\ D_1 + D_2 & \text{if } W_{li} \in \partial W_{li} \text{ and } C_r < 0 \end{cases} \quad (13)$$

where $D_1 = e_i S_{li}(\bar{u}_i)$, $D_2 = -D_1^T W_{li} ((1 + |W_{li}|)/w_{li})^2 W_{li}$, and $C_r = e_i S_{li}^T(u) W_{li}$.

These control and update laws guarantee the uniform ultimate boundedness with respect to the arbitrary small set:

$$\varepsilon_{c_i} = \left\{ e_{c_i}(t) : |e_{c_i}| \leq \frac{\varepsilon_i}{\gamma}, \gamma > 0 \right\}$$

The previous results practically state that if we start from inside the set ε_{c_i} then e_{c_i} are uniformly bounded by ε_i/γ . Otherwise there exists a finite time in which e_{c_i} reaches the boundary of ε_{c_i} and remains there in for all time thereafter. It is obvious that the performance of the continuous time control law is controlled by ε_i and γ . The first term ε_i is strongly related to the continuous time model accuracy, while γ is a design constant which can be chosen arbitrary. This practically means that no matter what the initial error, the proposed direct adaptive control scheme guarantees that all buffer states will approach their target values in finite time, always staying in close proximity to them.

Hence, this direct adaptive control scheme, ensures that all buffer states can in finite time approach their target values, and always stay in close proximity to them, and by further controller fine-tuning they can get arbitrarily close to their target. Obviously, when considering the output buffers of the manufacturing cell, it is clear that it has been proven that this approach can lead arbitrarily close to the production goal in finite time.

Real Time Scheduling

The control input signal obtained by the control law (Equations (8) through (13)) is a continuous frequency signal. Therefore, some dispatching policy has to be employed to determine the actual parts dispatching times.

For the single product case where only one submachine is allowed, the algorithm is as follows. The controller output is sampled at a certain time instant, and the corresponding operating frequency is transformed to a time interval by taking its reciprocal. At the same time instant a new command is sent to the specific submachine. Control and update laws are allowed to evolve in time while the submachine is left untouched until the precalculated time interval is completed. This interval is greater or equal to the machine operation time, since the control input frequency is less than or equal to u_{max} . Afterwards, a new sample

of the control input is taken and the process is repeated. Clearly, this control input discretization introduces some modeling error, to be discussed in the following subsection.

For the case of FMS systems containing multiproduct machines, in order to determine which buffer is to be served, the use of some criterion representing a measure of the priority to be given is proposed. For a certain machine m containing $N(m)$ submachines, the criterion is calculated on each route which employs any submachine $s \in O(m)$. The derivation of the criterion value is based on the semifinished product availability, (work-in-process, WIP), as well as on the control error of the route's output buffer. According to the proposed policy, the route to be served is the one with the largest criterion value. The proposed criterion value for each submachine s is given by:

$$J_s = \prod_{i=1}^{N_B(s)} f(x_i^-) \left[\lambda_1^- f(x_1^-) + \lambda_2^- f(x_2^-) + \dots + \lambda_{N_B(s)}^- f(x_{N_B(s)}^-) + \lambda_1^+ g(x_1^+) + \lambda_2^+ g(x_2^+) + \dots + \lambda_N^+ \frac{e^2}{1+e^2} \right] \quad (14)$$

where x_i^- , $i=1,2, \dots, N_B(s)$ are submachine s preceding buffer levels and x_i^+ $i=1,2, \dots, N$ are the levels of the buffers that follow on the same route, while N denotes the number of submachines that follow submachine s along its route including the output buffer.

The control error for the later is denoted by e . The parameters λ_i^- and λ_i^+ are weighting factors to be determined. The dependence of each argument on s has been omitted for the sake of simplicity.

Continuous Control Input Discretization

In the previous definition, $f(\cdot)$ is a positive monotonically increasing non-linear function, with $f(0)=0$ and $f(c)=1$, where c stands for the corresponding buffer capacity. Obviously this function outputs the largest value for the case of a large product accumulation on the feeding buffer. Similarly, $g(\cdot)$ is a mirrored version of $f(\cdot)$, with $g(0)=1$ and $g(c)=0$. This function outputs the maximum value if the following buffers are almost empty. In this way conditions like the feeding buffer is almost empty and/or the following buffers are in the vicinity of their maximum capacity, lead to small J_s which in turn means that the corresponding route will not be served. Functions $f(\cdot)$ and $g(\cdot)$ can be closely approximated by shifted sigmoids, as in Rovithakis, Perrakis, and Christodoulou (1997, 1999) to summarize the discrete dispatching decision algorithm in the multi product case is as follows:

Algorithm II.1 (Discrete Dispatching Decision-Multi Product Case)Input $u \in \mathcal{R}^B, x \in \mathcal{R}^B, e_c \in \mathcal{R}^B$

If machine is not FMS then

I.1 Controller output is sampled at a certain time instant.
A time interval equal to the reciprocal of the sample
is computedI.2 At the same time instant a new command is sent to the
specific submachine.

I.3 As soon as the time interval is completed, go to Step I.1

Else if the machine is FMS then

II. for all submachines that are not idling and conflict.

II.1 calculates J_s as in (14).II.2 selects the submachine with the largest J_s .III. Apply the appropriate command for the selected
submachine i.e., execute steps I.1, I.2 of the non-FMS
machine caseIV. When the currently working submachine finishes processing and
starts idling goto Step II

Discretization Effects

The continuous time controller developed previously, contains the actual scheduling as follows:

$$\underline{u}_i = u_{d_i} + \omega_i(\bar{x}_i, \bar{u}_i, u_i)$$

where u_i is the continuous time control law, u_{d_i} is the actual scheduling applied and $\omega_i(\bar{x}_i, \bar{u}_i, u_i)$ is the bounded difference between the aforementioned signals.

It can be shown (Rovithakis, Perrakis, & Christodoulou, 1999) that in this case the control error e_{c_i} possesses a uniform ultimate boundedness property with respect to the arbitrary small set:

$$\varepsilon_{c_i} = \left\{ e_{c_i}(t) : |e_{c_i}| \leq \frac{w_{0i}^+ |\omega_i(\bar{x}_i, \bar{u}_i, u_i)| + \varepsilon_i}{\gamma}, \gamma > 0 \right\}$$

The term $(w_{0i}^+ |\omega_i(\bar{x}_i, \bar{u}_i, u_i)| + \varepsilon_i)/\gamma$ serves as a performance index that can be improved mostly by allowing the design constant γ to admit larger values. Moreover, better approximation of the continuous control law by the actual scheduling will lead to smaller values of $\omega_i(\bar{x}_i, \bar{u}_i, u_i)$ which in turn will improve further the overall performance.

Hence the convergence and robustness properties of the controller is still retained, whatever definition of J_s .

Test Case: SHW Mechanical Workshop

To investigate, assess and establish the applicability and value of the proposed approach to the scheduling problem, it has been considered necessary to examine data taken from existing manufacturing systems, with all the constraints, and complexity inherent in real-world cases. Thus an existing German industry, producing machine tools, has been chosen in order to evaluate the algorithm's performance on production processes crucial for the company's prosperity.

The test case is derived from one of the oldest industrial enterprises in Germany, *Schwaebische Huettenwerke (SHW)*, and specifically from the mechanical workshop of its *Machine Tool Division*. Department products are tools ranging from simple milling machines to universal milling, boring, and turning centers, and in their vast majority are produced on demand. Since on demand production characterizes the machine tool division operation, production is organized according to orders arriving on the division. Hence the scheduling problem has been formulated by a representative, bulky subset of the entire set of orders processed over several months of workshop operation. Therefore, the SHW scheduling problem has been posed as the problem of production of 18 different order types demanding an equal number of product types. The different product type routes are shown in Table 1.

Manufacturing Cell Topology

All numbers under the label "Processing Routes" are identification numbers of mechanical workshop machines and the external to the workshop assembly points visited. In the

Table 1. Manufacturing-cell topology

Order	Processing Routes										
1331	737	753	773	704	737	704					
1414	753		773	704							
1623	753		773	704							
2409	753	999	773	704							
2420	753	999	999	736	704						
1953	731	704	999	773	739	737	704	775			
2422	773	783	704								
2685	783	704									
2766	708	731	774	739	783	783	774	704			
3057	999	753	999	773	775	999	737	999	999	704	
3061	753	966	773	736	704						
2694	728	732	722	783	704	783					
2783	732	752	731	777	999	777	731	722	783	783	704
2900	773	704									
2904	966	966	704								
1312	704	775	732	783	704						
2916	753	773	704								
3207	999	999	753	773	999	999	999	704			

Table 2. Processing steps duration

Order	Processing Routes										
1331	31.0	25.2	30.6	20.0	9.4	20.0					
1414	23.4	15.6	20.0								
1623	21.0	15.6	20.0								
2409	43.2	3.0	27.0	20.0							
2420	34.2	0.5	3.6	34.2	20.0						
2422	16.8	1.2	20.0								
2685	2.4	20.0									
2766	7.7	1179	244.2	348.0	46.8	25.2	111.6	20.0			
3057	20.0	37.2	20.0	46.0	20.0	20.0	14.4	61.2	90.0	20.0	
1953	162	20.0	20.0	119.0	72.0	65.0	20.0	20.0			
3061	22.8	153	18.2	45.0	20.0						
2694	876	43.4	990.0	36.0	20.0	1.8					
2783	14.2	351	444.0	8.4	20.0	8.4	1260	1539	10.8	12.6	20.0
2900	24.0	20.0									
2904	1.2	6.2	20.0								
1312	20.0	20.0	11.0	1.2	20.0						
2916	22.8	25.8	20.0								
3207	20.0	20.0	22.8	19.8	3.6	4.8	24.6	20.0			

tabulated set of part type routes, 18 different types of products are produced in this cell of 18 processing locations, following routes in which a part type may visit a machine more than once, while no assembly operations are involved. Each entry of the table represents a processing step of a specific duration. In Table 1, the (i, j) entry being equal to m , means that the i -th product in its j -th processing step, visits machine with label m and from the non-linear adaptive controller viewpoint each such distinct operation corresponds to a unique submachine.

Table 2 shows the duration of the processing steps. The respective submachine will be referred as $s_{i,j}$ and controls the state of a buffer attached to its output which will be noted as $x_{i,j}$. Furthermore, a neural adaptive controller is assigned to every submachine and the output signal of the controller which regulates the state of the $x_{i,j}$ buffer (i.e., the frequency signal controlling the operation of submachine $s_{i,j}$), will be denoted as $u_{i,j}$.

Also, the first submachine of every product route, $s_{i,1}$, $i = 1 \dots 18$ is preceded by a buffer that temporarily stores raw materials of the respective, i -th product, which will be referred as $x_{i,0}$. $L(i)$, $i = 1 \dots 18$ will denote the number of processing steps in route i , thus $x_{i,L(i)}$ will be the output buffer collecting the finished products of route i . Also, $x_{i(i,m)}$ will stand for the target state of buffer $x_{i,m}$, thus $x_{i(i,L(i))}$ will denote the demand for product type i . Any further information necessary for the complete definition of the scheduling problem will be presented with direct reference to the cell topology, as provided in Table 1.

The duration in minutes of all processing steps underlying the presented cell topology is given in Table 2, where the value $T(i, j)$ of its (i, j) element stands for the duration of the (i, j) processing step.

The scheduling problem will be completely defined, by specifying the desired target states, the capacity and the initial state values, for all buffers in the system, as well as the rates under which raw materials for the different part types arrive in the system.

Specifically, all intermediate buffers' target states have been set equal to one and their capacity buffers with the exception of buffers $x_{8,1}$, $x_{12,1}$, $x_{13,1}$ where three parts capacity are allowed.

All buffers are considered to be initially empty. Raw material buffers have a two-part capacity; and when this capacity is reached, arrival of raw materials is ceased until the raw material buffer level is decreased.

Thus, considering its potential aspects of difficulty, the formulated problem constitutes a reasonably complicated one, mainly due to the diversities and variability of part processing times in the various submachines, the inequalities in machine's workload and its dimension. It is a problem of challenging dimensionality since it involves 18 different final product types, visiting 18 machines and a total of 95 different operations take place, thus implying the existence of 95 submachines, for the neural network scheduling algorithm.

The most extreme differences exist for the 13th product type 2783, where the processing times in the 11 submachines visited range from 8.4 to 1539.0 minutes. Obviously, the equivalent problem, in the sense of our proposed scheduler, of efficiently regulating in real time the state of 95 buffers, will be a task far from trivial, justifying the selection of the SHW test case, as a means of ultimate neural scheduler verification.

In order to gain further understanding of the underlying neural network controller structure, a subset of the interconnections of the inherent submachine topology is described next. Following the notation introduced in (Rovithakis, Perrakis, & Christodoulou, 1996, 1997, 1999) let $O(m)$ denote the set of submachines $s_{i,j}$ of machine m and $N(m)$ its cardinality. Let $B_+(s)$ be the set of input buffers to submachine s , $B_-(s)$ be the set of output buffers of submachine s , and $M(b)$ the set of submachines fed by buffer b . For the specific test case we obtain: $B_+(s_{i,j}) = \{x_{i,j-1}\} \forall i, j$, $B_-(s_{i,j}) = \{x_{i,j}\} \forall i, j$

For example, close examination of the cell definition yields the set of submachines related to machine 773:

$$O(773) = \{s_{1,3}, s_{2,2}, s_{3,2}, s_{4,3}, s_{6,4}, s_{7,1}, s_{10,4}, s_{11,3}, s_{14,1}, s_{17,2}, s_{18,4}\}, \text{ thus } N(773) = 11.$$

Moreover, the sets that completely define the route of the 11th product type are given next.

Sets of input buffers:

$$B_+(s_{11,1}) = \{x_{11,0}\}, B_+(s_{11,2}) = \{x_{11,1}\}, B_+(s_{11,3}) = \{x_{11,2}\}, B_+(s_{11,4}) = \{x_{11,3}\}, B_+(s_{11,5}) = \{x_{11,4}\}$$

Sets of output buffers:

$$B_-(s_{11,1}) = \{x_{11,1}\}, B_-(s_{11,2}) = \{x_{11,2}\}, B_-(s_{11,3}) = \{x_{11,3}\}, B_-(s_{11,4}) = \{x_{11,4}\}, \\ B_-(s_{11,5}) = \{x_{11,5}\}$$

Finally, the sets of submachines fed by each buffer in route 11 are listed:

$$M_-(x_{11,0}) = \{s_{11,1}\}, M_-(x_{11,1}) = \{s_{11,2}\}, \quad M_-(x_{11,2}) = \{s_{11,3}\} \\ M_-(x_{11,3}) = \{s_{11,4}\}, M_-(x_{11,4}) = \{s_{11,5}\}, \quad M_-(x_{11,5}) = \{\emptyset\}$$

Taking up to fourth-order terms for each argument, the dynamic equations employed for the buffers of the 11th product route are:

$$\begin{aligned} \dot{x}_{11,1} &= W_{0,11,1}^*{}^T [S(x_{11,1})^T, S(u_{11,1})^T]^T u_{11,1} + W_{1,11,1}^*{}^T S(u_{11,2}) + \varepsilon(x_{11,1}, u_{11,1}, u_{11,2}) \\ \dot{x}_{11,2} &= W_{0,11,2}^*{}^T [S(x_{11,2})^T, S(u_{11,2})^T]^T u_{11,2} + W_{1,11,2}^*{}^T S(u_{11,3}) + \varepsilon(x_{11,2}, u_{11,2}, u_{11,3}) \\ \dot{x}_{11,3} &= W_{0,11,3}^*{}^T [S(x_{11,3})^T, S(u_{11,3})^T]^T u_{11,3} + W_{1,11,3}^*{}^T S(u_{11,4}) + \varepsilon(x_{11,3}, u_{11,3}, u_{11,4}) \\ \dot{x}_{11,4} &= W_{0,11,4}^*{}^T [S(x_{11,4})^T, S(u_{11,4})^T]^T u_{11,4} + W_{1,11,4}^*{}^T S(u_{11,5}) + \varepsilon(x_{11,4}, u_{11,4}, u_{11,5}) \\ \dot{x}_{11,5} &= W_{0,11,5}^*{}^T [S(x_{11,5})^T, S(u_{11,5})^T]^T u_{11,5} + \varepsilon(x_{11,5}, u_{11,5}) \end{aligned}$$

$$\text{with } S(x_{i,j}) = [s(x_{i,j}) \ s^2(x_{i,j}) \ s^3(x_{i,j}) \ s^4(x_{i,j})]^T$$

$$S(u_{i,j}) = [s(u_{i,j}) \ s^2(u_{i,j}) \ s^3(u_{i,j}) \ s^4(u_{i,j})]^T$$

$$S_1(u_{i,j}) = [s(u_{i,j}) \ s^2(u_{i,j}) \ s^3(u_{i,j}) \ s^4(u_{i,j})]^T$$

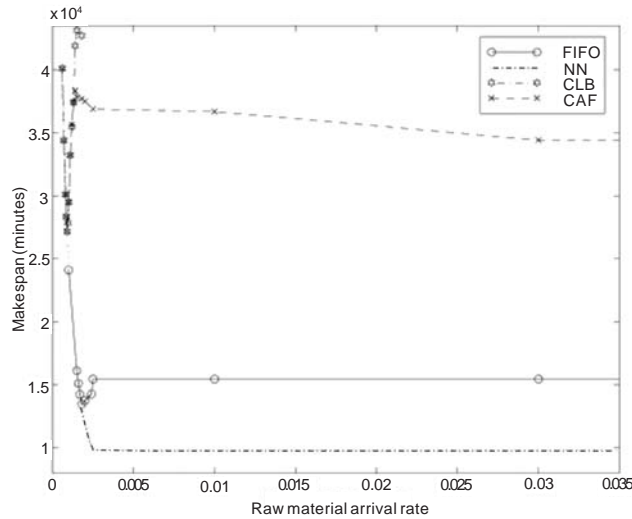
$$\text{where } W_{0,j}^* \in R^8 \ \forall j \in \{1,2,\dots,5\} \text{ and } W_{1,j}^* \in R^4 \ \forall j \in \{1,2,\dots,4\}$$

The NN model is further used in the development through Equations (8) to (13) of the continuous time control law $u_{11,j} \ j = 1, \dots, 5$

Conventional Scheduling Policies Used Modifications

A set of conventional scheduling methodologies has been selected for comparison purposes. The employed methods, real-time priority dispatch rules, distributed and of local scope, are the following:

Figure 1. Production makespan



- (1) First In, First Out (FIFO): Select submachine $s_{i,j}$ whose input buffer $x_{i,j-1}$, contains the part which arrived earlier than all parts contained in the input buffer of any other submachine in $O(m)$.
- (2) Clear Largest Buffer (CLB): Select $s_{i,j}^*$ such that $x_{i,j-1}^*(t) \geq x_{i,j-1}(t) \forall s_{i,j} \in O(m)$.
- (3) Clear a fraction (CAF): Select $s_{i,j}^*$ such that $x_{i,j-1}^*(t) \geq \varepsilon \sum_{j=1}^{N(m)} x_{i,j-1} \forall s_{i,j} \in O(m)$.

Clearing methods, CAF & CLB, process all parts of the selected buffer until it becomes empty, while FIFO processes a single part at a time.

It should be noted that the actually employed policies are variations of the previous definitions, which incorporate the following modifications, for stability reasons:

- (1) All dispatching commands for submachines whose output buffers have reached their capacity are ignored.
- (2) Selection is restricted to the set of submachines that participate in part type routes for which production has not been fulfilled.

The first modification concerns cancelling the application of the currently issued dispatching command for a specific submachine, whenever the corresponding buffer reaches its capacity limit. The second modification has been employed to overcome problems such as overall production paralysis or continuous processing of specific routes, by restricting production scheduling only to those product types for which production has not been fulfilled. Precisely, this modification adds a global perspective

to the criteria, since their original local scope and reliance on information only about jobs in the examined machine's queue becomes insufficient for the current problem.

For a more detailed justification of the use of these two modifications to the conventional scheduler the interested reader is referred to Rovithakis et al. (2001).

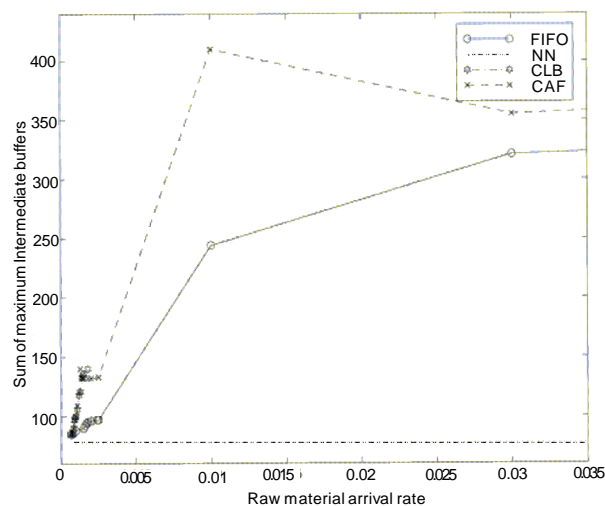
Results

Extensive simulation experiments that pursue the systematic extraction of the properties of the proposed scheduler, the investigation of its overall performance and validity and accuracy of the derived solutions' qualities, is the theme addressed in this section.

In what follows, we shall discuss the results obtained when demanding 15, 25, 20, 20, 25, 20, 20, 3, 10, 8, 15, 5, 2, 20, 20, 15, 10, 20 parts for the 18 product types respectively, beginning from clear intermediate and output buffers.

The evaluation study, adopts the commonly occurring in literature (Angsana & Passino, 1994) computation of a number of performance indices, which are mostly related with the time a job spends in the shop. Besides the yielded makespan that illustrates the rapidity in production goal achievement, indices of WIP, inventory, and backlogging cost have been selected, due to their popularity and the accuracy in investigating some of most primitive and important costs inherent in any manufacturing process. Measures of sum of maximum intermediate and output buffers states are included since they highlight schedulers' ability of maintaining stability with respect to capacity limitations while lead times are computed as average estimations of the delay for the production of a single part in a cell, which is invaluable information for higher levels of production planning decisions.

Figure 2. Sum of maximum intermediate buffer states



Makespan

More precisely, in Figure 1 the makespan, (i.e., the time period elapsing until the production goal is achieved for all product types), is plotted for various raw-material-arrival rates.

Examination of this set of curves reveals that for all raw material arrival rates greater than 0.0018 parts per minute, the proposed neural network scheduling methodology, results in significant speedup of the achievement of production demand. Considering the modified conventional schedulers, the achievement of production goal requires longer time intervals. Due to the potential instability of the selected policies, the inability of CLB to terminate is observed for rates greater than or equal to 0.002 parts per minute. Thus, in this and in all subsequent figures, CLB values are plotted only for the range of rates for which production goal is achieved (i.e., for rates less than 0.002). In the subsequent analysis the somewhat arbitrary definition that rates lower than 0.0027 are low is used, while rates greater than 0.0027 will be referred to as high. Hence, for low raw-material-arrival rates similar or even identical performance for all compared schedulers is observed. This originates from the fact that for lower raw-material-arrival rates, makespan is dominated by the time required for the entrance in the cell of all raw materials for the product types with the highest demand. It has been assumed that initially, all raw material buffers contain a single part. Since a maximum demand of 25 parts has been defined, and one part arrives every $1/rate$ minutes, it is easily derived that $24 \times 1/rate$ minutes must elapse before all necessary raw materials have entered the system.

For the case of high rates, it is necessary, for comparison purposes, to extract a lower bound for the corresponding makespan. An obvious one should be the maximum over all machines of the sum of the duration of the minimum work that should be done by all submachines of each machine, plus the minimum time required for a single part to reach

Figure 3. Sum of maximum output buffer states; both FIFO and NN-based scheduling algorithms coincide for all raw-material-arrival rates

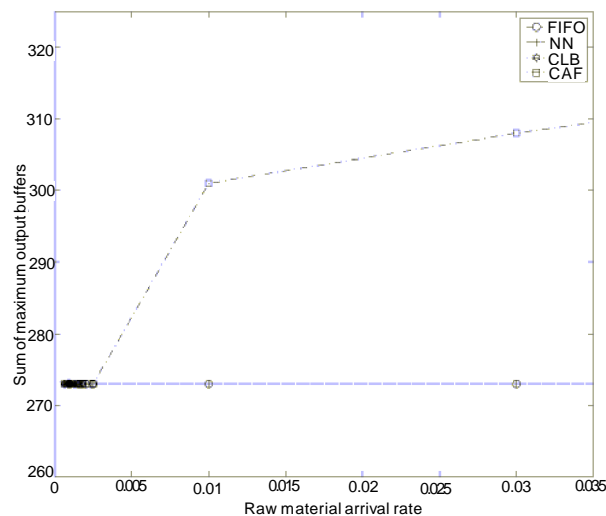
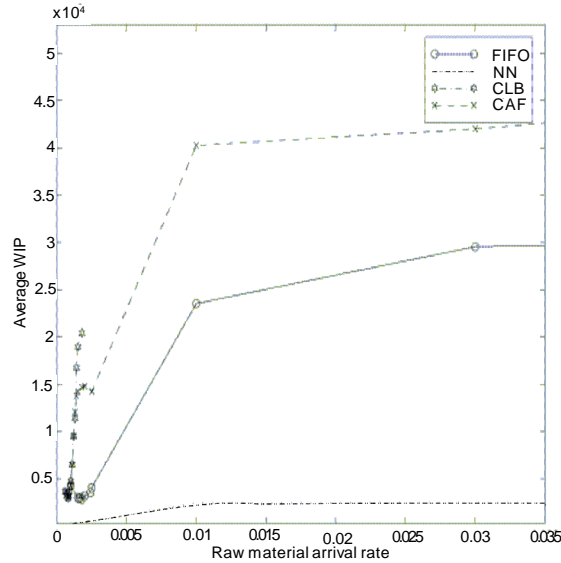


Figure 4. Average WIP cost



machine input. The derivation of this lower bound relies on the assumption that except for the first part to be processed in a machine, there are parts always available in the input buffers of its submachines, which is obviously optimistic for the majority of submachines in this complex cell. Thus, the previous maximum is calculated under the assumption that no machine is idling. The minimum work that any submachine in the cell should do is equal to the corresponding product demand. Thus, the time required for the minimum work of machine m is

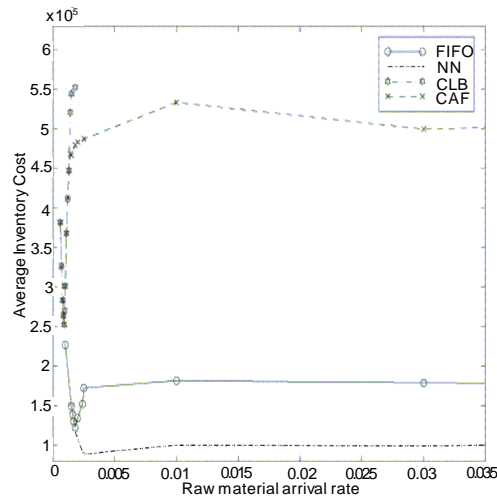
$$T_m = \sum_{s_{i,j} \in O(m)} T_{i,j} \times x_{r(i,L(i))}.$$

In the specific test case the lower bound has been found equal to 8947.3 min.

The neural network methodology achieves the production goal in 9778 minutes thus obtaining a deviation from the lower bound equal to 9.29% for raw-material-arrival rates greater than or equal to 0.0027. Meanwhile, optimal performance for the conventional schedulers FIFO, CAF and CLB occur at much lower rates, specifically for 0.0018, 0.0009 and 0.0009 parts per minute, where the lower bounds are provided by the formula $24 \times 1/\text{rate}$ with corresponding deviations 0.867%, 1.865%, 1.87% and makespan 13449, 27164 and 27165 minutes respectively.

Simulations with higher raw-material-arrival rates, which provide higher raw-material availability, resulted in no change of the makespan for both the neural network and FIFO schedulers, while analogous performance is yielded by CAF.

Figure 5. Average inventory cost



Sum of Maximum Intermediate and Output Buffer States

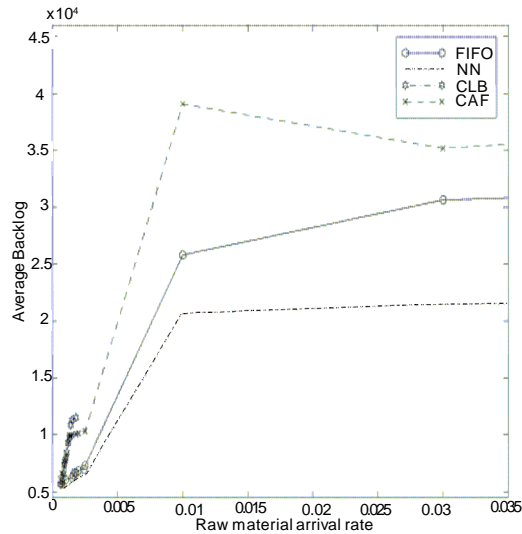
An overview of the peak buffer states experienced by the intermediate buffers is presented in Figure 2, using the sum of the maximum state of all intermediate buffers. For the case of neural network scheduler this sum is equal to the number of intermediate buffers in the cell (i.e., 77), since the algorithm assigns a target value of one for all intermediate buffers, which cannot be exceeded as the control law forces its output to become zero whenever a buffer state reaches its target value.

When the conventional policies are used, this sum becomes considerably larger and as can be seen from Figure 2, it can be more than five times higher as in the case of CAF. Specifically, while for very low rates, this sum slightly exceeds the limit of 77, a slight increase in the rate results in a considerable increase in the sum of the maximum states for intermediate buffers with most extreme case that of CAF at 0.001 parts per minute.

The NN scheduler controls robustly the maximum possible states of the buffers with respect to raw-material-arrival rates. This is obviously not the case for all conventional schedulers studied that, at higher rates, even force the buffers to reach their capacity limits. Therefore, the proposed scheduler ensures stable manufacturing system operation and minimum capacity requirements for buffers.

Figure 3 presents the sum of maximum output buffers states. The achieved results for both FIFO and NN are identical and equal to the total demand of 273 parts for all product types. This sum for the case of the rest of the schedulers, acquires larger values, thus denoting the production of a surplus of parts. This excess of parts originates in the clearing nature of the employed policies (i.e., CAF, since CLB is not functioning for rates greater than 0.002).

Figure 6. Average backloging cost



At low rates, all schedulers result in precise achievement of the production goal, due to the fact that all operations, even the clearing ones, empty buffers containing very few parts, causing submachines that output finished products, to process buffers which contain a single part only. Both FIFO & NN emerge as schedulers guaranteeing accurate achievement of production.

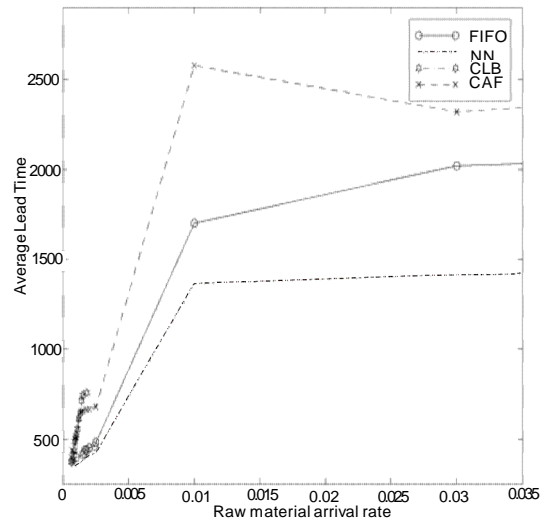
WIP and Inventory Costs

The next two figures represent cost measures for storing parts in buffers, intermediate and output respectively. Let the cost for holding $x_{i,j}$ parts at a buffer for T time units be considered equal to $k_{i,j} x_{i,j} T$, where $k_{i,j}$ is the cost for storing a single part in the output buffer of submachine $s_{i,j}$, per time unit. Integrating the previous quantity over the entire makespan yields the cost for storing parts in a single buffer. Thus, assuming for simplicity that for all buffers in the system $k_{i,j}=1$, the cost measure evolves into a plain integral of the respective buffer states. Specifically measures for the Work In Process (WIP) and Inventory costs, are provided by means of the average integral of the intermediate and

output buffer states respectively, (i.e. WIP equals to $(I/N_i) \sum_{i=1}^{18} \sum_{j=1}^{L(i)-1} \int_0^T x_{i,j} dt$, while

Inventory cost is $(I/N_o) \sum_{i=1}^{N_o} \int_0^T x_{i,L(i)} dt$ where $N_i=77$, $N_o=18$ are the number of intermediate and output buffers in the cell and T is the production makespan yielded by each scheduling methodology). Due to the low intermediate buffer states guaranteed by the neural network algorithm, a lower WIP is observed, while the rest of the schedulers introduce considerably larger costs.

Figure 7. Average lead time



Thus, the neural network algorithm achieves the most efficient WIP cost control, by keeping intermediate buffer state as small as desired, while guaranteeing system stability. Inventory costs are also considerably smaller than those occurring when employing the conventional scheduling policies due to the overall faster fulfilment of the production demand.

For lower rates, when all schedulers achieve total production almost simultaneously, deviations between the inventory costs yielded by the considered policies, and the one introduced by the NN are not really significant with the exception of the far worst performance of CLB. However, the increase in rates is causing the occurrence of a considerably larger peak in intermediate and output buffer states as well as significant differences in makespan, which sufficiently justify, the observed extremely greater WIP & inventory cost for the case of the conventional schedulers, shown in Figures 4 and 5. For higher rates the NN scheduler presents a far more robust-stable behavior when compared to FIFO and CAF. Thus, NN superiority with respect to these popular cost measures is obvious.

Backlogging Cost and Lead Time

Considering backlogging costs, (i.e., the cost resulting from the delay in achieving production targets), the NN scheduler reaches production goal with the minimum cost when compared to the rest of the examined policies. Specifically, the employed backlogging cost has been defined as follows: $(1/N_o) \sum_p \Psi_p(t) = (1/N_o) \sum_p \int_0^t \beta_p(\tau) d\tau$ where $\beta_p(t) = I_p(t) - O_p(t)$ with $I_p(t)$, $O_p(t)$ the number of parts of type p that have entered and exit the cell until time t .

As Figure 6 demonstrates, backloging costs remain small for relatively low rates, since buffers in the system contain few parts, less conflicts among submachines occur and thus finished products are outputted with reduced delays. On the other hand, increase in raw-material-arrival rates, leads to overall increased buffer levels and conflicts, thus making the output of finished products to occur with a considerably larger delay. For any rate, the NN constitutes a considerably less expensive methodology (with respect to backloging cost) than all examined policies.

Finally, Figure 7 gives the average lead time, (i.e., the average of the time elapsing between the input of a raw material in the cell and the output of the respective finished product), for all product types and all parts produced in the cell. Practically, lead time is equivalent to the widely used cost index of average *flowtime* (Silver, Pyke, & Peterson, 1998). Comparing this plot with backloging cost curves, an impressive similarity in slope and shape is immediately observed. Actually all reasoning about the causes of backloging cost, also apply for the case of product lead times, where the superiority of the NN algorithm is once again justified by the smaller production makespan.

Considering implementation issues, these simulations justify the claim that the NN scheduler constitutes a real time algorithm. Simulation of almost 10.000 minutes of cell operation lasted about 20 minutes on a Solaris workstation, featuring a Pentium II/266 MHz processor. Computation of the control vector $u(t)$ given the current buffer states, demanded 0.1616 seconds of CPU time. Moreover, a hardware implementation of the algorithm is expected to be characterized by negligible computational delays, yielding a distributed real-time nonmyopic robust controller structure.

Conclusion

In this chapter our scheduling methodology previously proposed in Rovithakis, Perrakis, and Christodoulou (1996, 1997, 1999, 2001) has been thoroughly presented. A challenging real world manufacturing cell has been considered and for a wide range of raw-material-arrival rates presented superb performance when compared with a set of conventional schedulers. With the exception of extremely low rates, where all schedulers converge to an almost identical performance in the remaining range of higher rates, the proposed algorithm features superb stability and robustness as well as efficient control of critical costs such as WIP, inventory and backloging, outperforming all the conventional schedulers discussed.

Thus, the previous analysis establishes the proposed neuroadaptive architecture's features of robustness and its potential of efficiently solving general, moderate sized and of arbitrary complexity dynamic, deterministic job shop problems. The appealing properties of the algorithm should be considered in conjunction with its real time property that enables immediate derivation of scheduling decisions and their application in the active manufacturing processes. The merits of the algorithm include the following features: (a) the capability of immediately incorporating demand alterations and thus instantly redefining targets for the appropriate buffers and its dimensionality, and (b) its size that grows linearly with the size of the examined scheduling problem, as one additional DNN

controller is demanded for every new distinct operation type, that is, every new spot of manufacturing activity in the production cell.

The presented methodology possible applications for the task of production scheduling range from employment as a simulation tool determining dispatch lists for shop floor on regular time intervals, to hardware implementation, where the latter would be obviously characterized by the capability of efficiently handling scheduling problems considerably larger in dimension and complexity than the current test case.

Enhancing our developed methodology may involve investigation of alternative model developments and thus appropriate control and update laws should be derived. Extending current framework in order to handle further manufacturing systems categories may result in several interesting issues. Thus, extending the methodology to include disassembly operations is a modest goal of future work, while enhancing current theory to allow for operation dependent setup time present in alternation in submachine operation may also be an interesting aspect. Moreover, theory enhancement such that bounded variabilities are allowed for machine operation times, without affecting the current robustness properties of the algorithm emerges as a challenging subject.

Furthermore, since the existing model is based on continuous time model, it could readily facilitate continuous flow systems where the buffer contents are continuously modified by fractional continuous flow of material at each individual time instant that the respective submachine is operating. The encouraging success of the present results, propose as a major research challenge, the investigation of ways of coupling discrete event dynamic systems (DEDS) with adaptive control techniques and adoption of RHONN structures as plant modeling mechanisms. It is hoped that such an approach if feasible, may facilitate inherently more accurate models while the yielded control system may be characterized by the attractive properties of real time adaptability, stability, and robustness.

In conclusion, the evaluated methodology constitutes a novel one, featuring real time operation together with the guarantees of stable and robust operation in the presence of any source of disturbances where the qualities of the resulting schedules establish it as a promising alternative to job shop scheduling in particular and in production scheduling in general.

References

- Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job-shop scheduling. *Management Science*, 34(3), 391-401
- Angsana, A., & Passino, K. M. (1994). Distributed fuzzy control of flexible manufacturing systems, *IEEE Transactions on Control Systems Technology*, 2(4).
- Blazewicz, J., Domschke, W., & Pesch, E. (1996). The job-shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93, 1-33.

- Brandimarte, P., & Villa, A. (1995). *Advanced models for manufacturing systems management*. CRC Press.
- Brucker, P. (1995). *Scheduling algorithms*. Berlin: Springer.
- Cheng, Runwei, Gen, Mitsuo, Tsujimura, Yasuhiro. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, Part II: Hybrid genetic search strategies. *Computers and Industrial Engineering*, 36, 343-364
- Fox, M. S., Allen, B. P., Smith, S. F., & Strohm, G. A. (June 1983). *ISIS: A constraint-directed reasoning approach to job shop scheduling: System summary* (CMU-RI-TR-83-3-8). Pittsburgh, PA: Carnegie-Mellon University, the Robotics Institute.
- Gershwin, S. B., Hildebrant, R. R., Suri, R., & Mitter, S. K (1986). A control perspective on recent trends in manufacturing systems. *IEEE Control Systems*, 3-15
- Glover, F. (1989). Tabu search-part I. *ORSA Journal on Computing*, 1(3), 190-206
- Grabot, B., & Geneste, L. (1994). Dispatching rules in scheduling: A fuzzy approach. *International Journal of Production Research*, 32(4), 904-915
- Gupta, Y. P., Evans, G. W., & Gupta, M. C. (1991). A review of multicriteria approaches to FMS scheduling problems. *International Journal of Production Economics*, 22, 13-31
- Hale, J. K. (1969). *Ordinary differential equations*. New York: Willey-InterScience.
- Ioannou, P. A., & Sun, J. (1996). *Robust adaptive control*. Upper Saddle River, NJ: Prentice-Hall.
- Jackson, J. R. (1963). Job shop like queuing systems. *Management Science*, 10(1), 131-142.
- James, S. (October 1996). A renaissance for process manufacturers. *APICS-The Performance Advantage*, 46-51.
- Kim, S. Y., Lee, Y. H., & Agnihotri, O. (1995). A hybrid approach for sequencing jobs using heuristic rules and neural networks. *Production Planning and Control*, 6(2), 445-454
- Kosmatopoulos, E. B., Polycarpou, M. M., Christodoulou, M. A., & Ioannou, P. A. (1995). Higher order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, (6), 422-431
- Kusiak, A. (1987). Designing expert systems for scheduling of automated manufacturing. *Ind. Eng*, 42-46
- Lawler, E. L., Lenstra, J. K., Rinnooy, K., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbook in Operations Research and Management Science*, 4, *Logistics of Production and Inventory*.
- Lourenco, H. R. D. (1995). Job-shop scheduling: Computational study of local search and large step optimization methods. *European Journal of Operational Research*, 83, 347-364.
- Morton, T. E., & Pentico, D. W. (1993). *Heuristic scheduling systems*. New York: Wiley.
- Panwalkar, S. S., & Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, 25(1), 45-61.

- Rodammer, F. A., & White, K. P. (1988). A recent survey of production scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6), 841-851
- Rovithakis, G. A., & Christodoulou, M. (1994, March). Adaptive control of unknown plants using dynamical neural networks. *IEEE Transactions on Systems, Man and Cybernetics*, 24(3).
- Rovithakis, G. A., & Christodoulou, M. (1995). A direct adaptive regulation of unknown nonlinear dynamical systems via dynamic neural networks. *IEEE Transactions on Systems Man and Cybernetics*, 25(12), 1578-1594.
- Rovithakis, G. A., & Christodoulou, M. A. (1997). Neural adaptive regulation of unknown nonlinear dynamical systems. *IEEE Transactions on Systems Man and Cybernetics*, (27), 810-822.
- Rovithakis, G. A., Gaganis, V. I., Perrakis, S. E., & Christodoulou, M. A. (1996, December). A recurrent neural network model to describe manufacturing cell dynamics. In *Proc. IEEE Conf. Decision Control*, Kobe, Japan.
- Rovithakis, G. A., Gaganis, V. I., Perrakis, S. E., & Christodoulou, M. A. (1997, December). Dynamic neural networks for real time control of FMS. In *Proc. IEEE Conf. Decision Control*, San Diego, CA.
- Rovithakis, G. A., Gaganis, V. I., Perrakis, S. E., & Christodoulou, M. A. (1999). Real-time control of manufacturing cells using dynamic neural networks. *Automatica*, 35, 139-149.
- Rovithakis, G. A., Perrakis, S. E., & Christodoulou, M. A. (2001). Application of a neural-network scheduler on a real manufacturing system. *IEEE Transactions on Control Systems Technology*, 9(2), 261-270
- Sabuncuoglu, I., & Gurgun, B. (1996). A neural network for scheduling problems. *European Journal of Operational Research*, 93, 288-299.
- Schonberger, R. J. (1983). Application of single-card and dual-card kanban. *INTERFACES*, 13(4), 56-57.
- Silver, E. A., Pyke, D. F., & Peterson, R. (1998). *Inventory management and production planning and scheduling*. Wiley.
- Van Laarhoven, P. J. M., Aarts, G. L., & Lenstra, J. K. (1988). *Job-shop scheduling by simulated annealing* (Report OS-R8809). Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica.
- Yamada, T., & Nakano, R. (1996). Job shop scheduling by simulated annealing combined with deterministic local search. In *Meta-heuristics: Theory and applications* (pp. 237-248). Hingham, MA: Kluwer Academic Publishers.

Chapter XV

Recognition of Lubrication Defects in Cold Forging Process with a Neural Network

Bernard F. Rolfe, Deakin University, Australia

Yakov Frayman, Deakin University, Australia

Georgina L. Kelly, Deakin University, Australia

Saeid Nahavandi, Deakin University, Australia

Abstract

This chapter describes the application of neural networks to recognition of lubrication defects typical to industrial cold forging process. The accurate recognition of lubrication errors is very important to the quality of the final product in fastener manufacture. Lubrication errors lead to increased forging loads and premature tool failure. Lubrication coating provides a barrier between the work material and the die during the drawing operation. Several types of lubrication errors, typical to production of fasteners, were introduced to sample rods, which were subsequently drawn under both laboratory and normal production conditions. The drawing force was measured, from which a limited set of statistical features was extracted. The neural-network-based

model learned from these features is able to recognize all types of lubrication errors to a high accuracy. The overall accuracy of the neural-network model is around 95% with almost uniform distribution of errors between all lubrication errors and the normal condition.

Introduction

Cold forging includes many processes such as bending, cold drawing, cold heading, coining, extrusion, punching, and thread rolling to produce a diverse range of part shapes. These include various shaft-like components, cup-shaped geometry parts, hollow parts with stems and shafts, all kinds of upset (headed) and bent configurations, as well as combinations of these geometries. The temperature of metals being cold forged may range from room temperature to several hundred degrees.

Often chosen for integral design features, such as built-in flanges and bosses, cold forging is frequently used in automotive steering and suspension parts, antilock-braking systems, hardware, defence components, and other applications where high strength, close tolerances and volume production makes it an economical choice.

In the cold forging process, a chemically lubricated slug is forced into a closed die under extreme pressure. The unheated metal thus flows into the desired shape.

Upsetting, or heading, a common technique for making fasteners, gathers steel in the head and other sections along the length of the part. In upsetting, the metal flows at right angles to the ram force, increasing the diameter and reducing the length.

A typical fastener manufacturing process uses batch production material transfer. The plant is divided into three main areas:

- Preprocessing that involves descaling and application of lubrication consisting of the zinc phosphate carrier and a soap stearate lubricant coating;
- Primary processing that involves wire drawing and extrusion (cold forging);
- Postprocessing that involves cleaning, heat treatment, and the application of a protective coating.

The lubrication used during preprocessing has a major impact on the productivity of the primary processing area. For example, if preprocessing fails to produce high-quality coated rod or the coating is damaged during the material handling then the output efficiency of the primary processing is decreased. This is a result of increased forging loads and premature tool failure, as well as increased defect sorting and the reprocessing of the coated rod. The lubrication coating must provide a barrier between the work material and die during the drawing operation, while still being sufficiently robust to remain on the wire during the transfer to the extrusion operation, where the wire undergoes multistage deformation without the application of additional lubrication.

This chapter describes the application of neural networks to the recognition of lubrication defects typical to an industrial cold-forging process employed by fastener manufacturers. The accurate recognition of lubrication errors, such as coating not being applied properly or damaged during material handling, is very important to the quality of the final product in fastener manufacture.

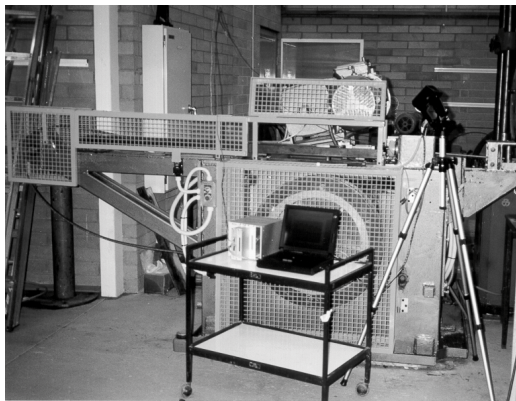
Background

The evaluation of the coating performance is done in industry usually through production-based methods. The main measures for coating performance are tooling changeover rates, and the detection of score marks appearing on drawn and forged surfaces. These production-evaluation techniques, while being valuable long-term indicators of coating performance, are reactive methods and are unable to assess the coating condition before it enters the primary processing area. This leads to tooling and product damage.

The evaluation technique developed at the School of Engineering and Technology, Deakin University (Savage, Kershaw, & Hodgson, 1999) uses a process-simulation test rig and a selection of analysis tools to evaluate the coating performance (Figure 1). This technique allows lubrication evaluation to be performed in isolation to production, enabling analysis to be done without interfering with day-to-day running of the production line.

The main performance requirements for the lubrication coating in fasteners manufacturing are from the preforge wire-drawing operation and the extrusion stages in the cold forging process. In the developed process-simulation test-rig, multi-reduction drawing simulates these stages in the fastener manufacturing process. The first reduction simulates the predrawing process while a second reduction simulates the extrusion stage of the cold forging process (Savage et al., 1999). The test rig was constructed from a variable-speed, continuous-drawing machine where quick-change multireduction die

Figure 1. The multidraw test rig



hostings, a quick-change rod-clamping mechanism, and a constrained flatbed were designed and installed. Strain gauges were mounted on the rod-clamping mechanism to detect drawing force at both the first and second reductions. Force signals are collected and conditioned by National Instruments hardware and Labview software. In this work, we deal with preforge wire drawing operation only.

Experimental Set-Up

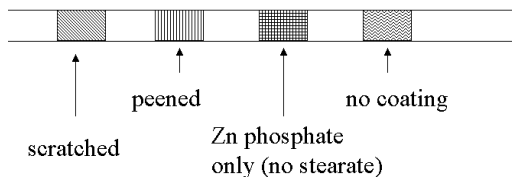
Laboratory Testing

A two-layer solid lubricant system was used on the rods. This was applied in a plant environment with conditions kept as close as possible to those used for standard production. Fifty samples were produced with a two-layer coating applied: zinc phosphate carrier and calcium stearate lubricant coating. The rods are pickled to clean them and a zinc-phosphate layer is deposited followed by a calcium-stearate layer. Another 20 samples were produced as before but with an additional coating of soap lubricant. This final coat of powdered lubricant was added to minimize the damage of the sensor's drawing die on the production wire the same way as it is done in plant.

Four different kinds of defects common in production of fasteners were introduced into the coatings (Figure 2):

1. No coating, where heat shrink-wrap was applied to rods prior to all steps in the coating process. This corresponds to missing coatings from a preprocessing stage;
2. Zinc phosphate only, where heat shrink-wrap was applied after the zinc-phosphate application. This corresponds to the missing calcium-stearate layer coating from a preprocessing stage;
3. Hammer peening of the surface of the bar. This type of error simulates defects introduced during material handling from preprocessing area to the primary processing;

Figure 2. Sample rod for the laboratory trial



4. Scratching of the coating by its removal parallel to the bar. This was introduced by filing the coating off. This type of error simulates coils being dragged across the shop floor during transfer from preprocessing area to the primary processing.

All defects were approximately 50mm in length and applied to the circumference of the rod with defects being separated by variable fully-coated lengths of rod.

The experimental test rig was used to produce the rod samples drawn with a 0.35 mm reduction from the starting diameter of 0.50 mm. The samples were drawn with an area reduction of approximately 7%, and the loads on the drawing die were monitored by strain gauges on the rod-clamping mechanism. All defects resulted in increased drawing loads. In the case of the hammer peening, this is likely to be due to the resulting irregularity of

Figure 3. Typical nonlubricated rod sample for the laboratory trial

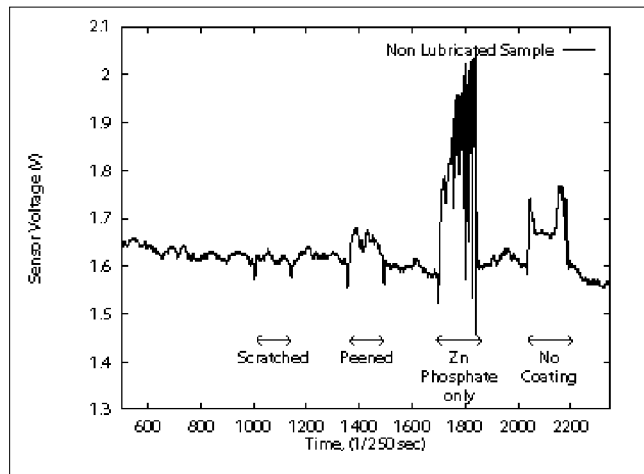
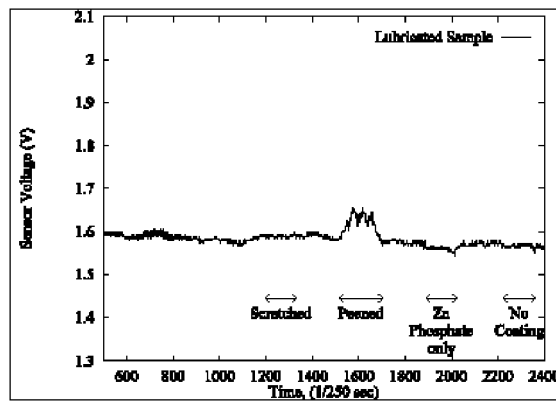


Figure 4. Typical lubricated rod sample for the laboratory trial



the rod diameter. In the other three cases, reduced efficiency of the coatings is due to missing lubrication components. The defect with only zinc phosphate layer resulted in the highest friction. The zinc phosphate is a soft coating and thus is likely to produce a galling or sticking effect as the rod passes through the die.

The typical force signatures for the rods with two layers of lubricants (labelled as nonlubricated samples) and for the rods with an extra layer of soap lubricant applied (labelled as the lubricated samples) are shown in Figures 3 and 4.

As can be seen, Error 4 (scratching of the coating) is visually indistinguishable from the normal condition (the one without any errors) on nonlubricated data, and only Error 3 (peening of the surface of the bar) is readily distinguished from the normal condition on the lubricated data.

The force signatures from these 70 trials were collated to create two time series, one for nonlubricated samples, and another for lubricated samples with all five possible lubricant conditions (normal condition and the four defects) appropriately prelabelled with a corresponding condition label being manually applied to each time step of the drawing force signal.

Production Testing

A two-layer solid lubricant system was used on the rods same as with laboratory testing. This was applied in a plant environment to a continuous wire as part of the standard production. The rod samples were drawn with a 0.008 mm reduction from a starting diameter of 1.400 mm.

Two plant trials were made with two-layer coating applied: zinc-phosphate carrier and calcium-stearate lubricant coating (Figure 5). Another two plant trials were done as before but with an additional coating of soap lubricant (Figure 6). This final coat of powdered lubricant was added to minimize the damage of the sensor's drawing die on the production wire.

Figure 5. Nonlubricated wire used in production trials

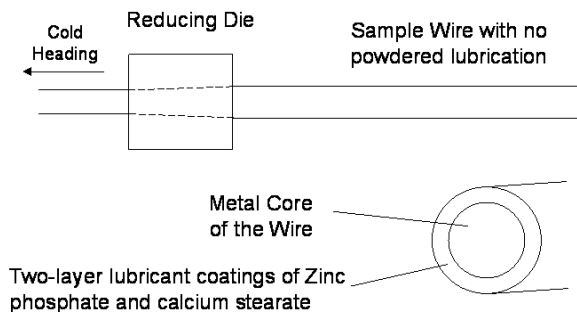


Figure 6. Lubricated wire used in production trials

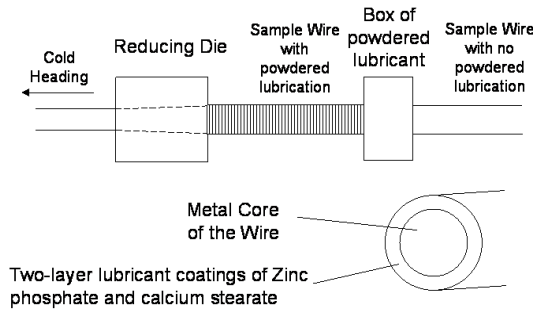
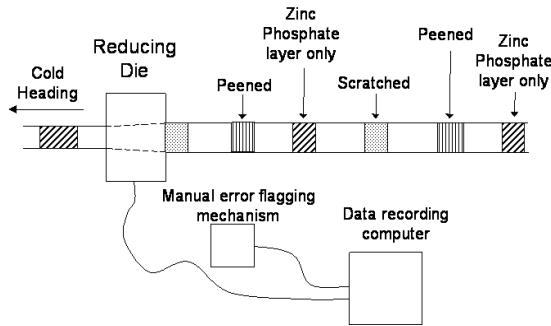


Figure 7. Experimental set-up for production trials



Three different kinds of defects common in production of fasteners were introduced into the coatings (Figure 7):

1. Zinc phosphate only, where heat shrink-wrap was applied after the zinc-phosphate application
2. Hammer peening of the surface of the bar
3. Scratching of the coating by its removal parallel to the bar

No coating defect that was introduced in the laboratory testing was not evaluated in the production trials due to a potential tooling and sensor damage.

The defects were varied in length and in the order of appearance in contrast to the laboratory trial and applied to the circumference of the rod with defects being separated by variable fully-coated lengths of rod.

The samples were drawn with an area reduction of approximately 0.54% and a load cell monitored the loads on the drawing die. All defects resulted in increased drawing loads

Figure 8. Typical nonlubricated wire sample in production trials

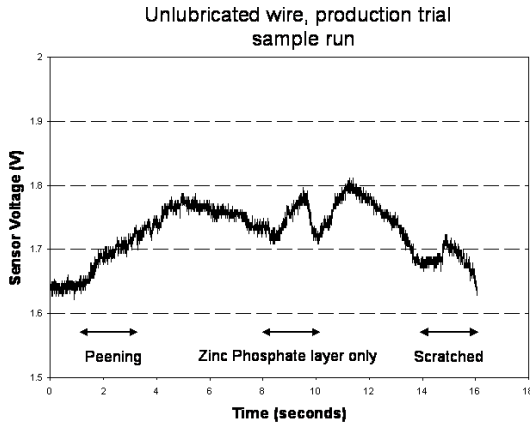
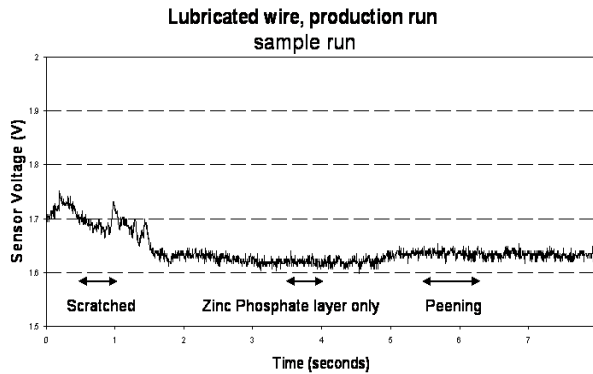


Figure 9. Typical lubricated wire sample in production trials



similar to laboratory trial. The defect with only a zinc-phosphate layer resulted in the highest friction similar to laboratory trial.

The typical force signatures for the rods with two layers of lubricants (labeled as nonlubricated samples) and for the rods with an extra layer of lubricant applied (labeled as the lubricated samples) are shown in Figures 8 and 9.

As can be seen, Error 3 (peening of the surface of the wire) is visually indistinguishable from the normal condition (the one without any errors) on nonlubricated data, and only Error 4 (scratching of the coating) is readily distinguished from the normal condition on the lubricated data.

The force signatures from these four plant trials were again collated to create two time series, one for nonlubricated samples, and another for lubricated samples with all four possible lubricant conditions (normal condition and the three defects) appropriately pre-labeled with a corresponding condition label to each time step of the drawing-force signal based on recording from a manual-error flagging (Figure 7). The data logging of lubrication defects was activated manually at the start and the end of each error condition. This resulted in the just approximate recordings of the start and the end of the error sequence in contrast to laboratory data where this information was more accurate.

Methodology

The main aim of this work is to develop an inductive model to identify accurately the lubrication errors in cold-forging by analyzing a force signature. That is, force variations from the nominal values are to be linked to various lubrication defects. This type of problem is generally called condition monitoring and uses advanced technologies in order to determine equipment condition and potentially predict failure. It includes, but is not limited to, technologies such as:

- Vibration measurement and analysis (e.g., Zheng & McFadden, 1999)
- Infrared thermography (e.g., Mattsson, Hellman, & Ljungberg, 2001)
- Oil analysis and tribology (e.g., De A Leao, Jones, & Roylance, 1996)
- Ultrasonics (e.g., Oh, Shin, & Furgason, 1994)
- Motor Current Analysis (e.g., Pöyhönen, Negrea, Arkkio, Hyötyniemi, & Koivo, 2002)

There are several possible approaches to the task of accurately identifying lubrication errors by analyzing the force signature. The most common approach is to formulate the problem as a classification (pattern recognition) problem (Breiman, Friedman, Olshen, & Stone, 1984).

We are interested in separating accurately the normal condition from all types of lubrication defects. However, a single drawing force value reveals very little information about what sort of condition the process is in. Extracting some useful statistical features from a set of contiguous force values can alleviate this limitation.

$$f_i = P_j(x_i, \dots, x_{i+n}),$$

where f_i is the feature, P_j is the j^{th} feature extractor function, x_i is the first force value of the set, x_{i+n} is the last force value of the set, and n is the size of the set.

The set of force values is defined as a sliding window. Associated with each sliding window is the lubrication condition or the output class:

$$h(f_1, \dots, f_m) \in (\text{normal}, \text{err1}, \dots, \text{err4}),$$

where $h(\cdot)$ is the lubrication condition and f_{1, \dots, f_m} are the extracted features of the window.

Therefore, we can associate each sliding window and its features with a corresponding lubrication condition. The true lubrication condition was chosen to be the output class associated with each sample point, x_{i+n} of the sliding window. In this case, the sliding window always extracts features from the past drawing force data. The size of a sliding window was 300 time steps and selected based on the minimal root-mean-square error between the predicted condition by the model and the actual condition of the lubricant.

The usefulness of the features extracted from the drawing force signal was analyzed by sensitivity analysis to evaluate their impact on the recognition rate for each lubrication condition and only the most important ones were retained.

The following statistical features were selected:

- Maximum force value within a sliding window;
- Minimum force value within a sliding window;
- Arithmetic mean (average) force value within a sliding window;
- Geometrical mean (median) force value within a sliding window;
- Standard deviation of the force values within a sliding window. This is a measure of how widely force values are dispersed from the average force value (the mean);
- Average deviation of the force values within a sliding window. This is a measure of the variability in a data set. It is the average of the absolute deviations of force data points from their mean;
- Skewness of the force values within a sliding window. This characterizes the degree of asymmetry of a force data distribution around its mean;
- Kurtosis of the force values within a sliding window. This characterizes the relative peakedness or flatness of a force data distribution compared with the normal distribution;
- Correlation between the force values within the current and the previous sliding windows;
- Covariance between the force values within the current and the previous sliding windows. This is the average of the products of deviations for each force data-point pair.

In addition to extracted features, we also used the current force signal and the past force signals with different time delays ranging from 25 time steps to 400 time steps also selected based on the minimal root-mean-square error between the predicted condition by the model and the actual condition of the lubricant. This resulted in 21 variables being available as inputs to the model: 10 statistical features as above, the measured force signal and 9 past force signal with different time delays as above.

Alternative methods for feature extraction used in condition monitoring include spectral analysis using Fast Fourier Transforms (FFT) (Kumar, Ravindra, & Srinivasa, 1997) and Wavelet Analysis (Luo, Osypiw, & Irle, 2003). However, our experiments have indicated that statistical features extracted from the force signal provided more information to distinguish between the normal condition of a lubricant and lubrication errors.

The outputs of the model were in normal condition (no lubrication defects), and all types of lubrication defects that were for laboratory data: no coating, zinc-phosphate layer only, peening and scratching, and for production data: zinc-phosphate layer only and peening and scratching. No defect representing no coating was introduced in the plant environment, due to possible damage to tooling and sensors in the plant.

As an inductive model within a pattern recognition framework we used a feed-forward multilayer perceptron (MLP) with backpropagation learning algorithm with a momentum term.

An attractive feature of an MLP network is that, given the appropriate network topology and the rigorous training procedures, they are capable of reliably characterizing a nonlinear functional relationship (Ripley, 1996). We used a hyperbolic tangent as the activation function of the MLP model. A pattern (online) learning and early stopping was employed. All input variables were normalized with zero-mean-unit average normalization.

The data used for modeling was obtained by data logging at the rate of 250 samples per second-strain gauges on the rod-clamping mechanism. As mentioned in the Experimental Set-up section, for the laboratory data these force signatures were collated to create two time series, one for nonlubricated samples and another for lubricated samples with all five possible lubricant conditions (normal condition and the four defects) appropriately prelabelled with a corresponding condition label being manually applied to each time step of the drawing force signal. For the production data, the force signatures resulted in two time series (as continuous rods were used), one for nonlubricated samples and another for lubricated samples with all four possible lubricant conditions (normal condition and the three defects) appropriately prelabeled with a corresponding condition label to each time step of the drawing force signal based on activating data logging of lubrication defects manually at the start and the end of each error condition. This resulted in the just approximate recordings of the start and the end of the error sequence in contrast to laboratory data where this information was more accurate.

As a result, four time series were obtained with 64,260 samples for nonlubricated laboratory trials, 51,140 for lubricated laboratory trials, 60,143 for lubricated production trials, and 64,736 for lubricated production trials. These four time series were split in two parts: first 70% of the data tuples were used for training and validation, and the remaining 30% were used to test the model generalization ability. An oversampling of the lubrication defect data points for training data was utilized to create an equal distribution of training-

Table 1. Model-prediction results for each of the lubrication conditions for the laboratory trial

Lab Trial	Nonlubricated samples					Lubricated samples				
Lub condition	NC	Err1	Err2	Err3	Err4	NC	Err1	Err2	Err3	Err4
Recognition rate	93.6 %	99.5 %	99.5%	98.0%	99.7%	94.4 %	99.3 %	99.6 %	97.8 %	99.1 %
Confusion matrices	13,407	8	4	21	3	11,090	6	3	11	7
	109	1,497	0	0	0	253	878	1	0	0
	68	0	1,362	0	0	131	0	895	3	0
	226	0	3	1,076	0	69	0	0	977	0
	510	0	0	1	983	209	0	0	8	801

Table 2. Model prediction results for each of the lubrication conditions for the plant trial

Plant Trial	Nonlubricated samples				Lubricated samples			
Lub condition	NC	Err2	Err3	Err4	NC	Err2	Err3	Err4
Recognition rate	97.4 %	98.4 %	97.2 %	98.1 %	94.6 %	98.8 %	99.6 %	99.8 %
Confusion matrices	10,277	31	69	21	11,110	33	11	5
	74	2,338	0	29	128	2,770	0	0
	169	6	2,416	0	360	0	2,680	0
	36	1	0	2,576	145	0	0	2,179

data tuples for each lubrication condition to avoid the problem of small disjunct (Frayman, Ting, & Wang, 1999; Weiss & Hirsh, 2000) as the data for the normal condition of lubrication (the one without any defects) dominates the available data tuples (being around 75% of all available data). However, the testing of the model generalization ability was performed on unmodified data samples.

The optimal parameters of the MLP were selected based on preliminary experiments. The parameters that resulted in the smallest root-mean-squared-error between the predicted lubrication conditions and the actual lubrication conditions were used. The selected MLP model consisted of 21 inputs and 5 outputs for the laboratory data (4 outputs for the production data) with two hidden layers; the first hidden layer has 50 nodes, and the second hidden layer has 45 nodes. The learning rate selected was 0.05; the momentum term was 0.99.

Results and Discussion

The overall prediction results of the MLP model and the corresponding confusion matrices for both laboratory and plant trials are in Tables 1 and 2. From Tables 1 and 2, it is clear that the performance of the MLP model is extremely accurate. The MLP model is able to distinguish almost perfectly the boundaries between all the errors for both

laboratory and production trials and only has some difficulty with recognition of the boundaries between the normal condition and the errors. Even these boundaries are recognized with a very high accuracy between 93% and 97%.

In Table 1, NC represents normal condition, Err1 represents no coating, Err2 represents zinc phosphate layer only, Err3 represents peening, and Err4 represents scratching. Confusion matrices here are 5 by 5 matrices showing how many samples belonging to a particular condition were classified accurately and how many samples were misclassified as other conditions, for example 13,407 nonlubricated samples were classified accurately as NC, 109 as Err1, 68 as Err2, 226 as Err3, and 510 as Err4.

Most importantly, while the extra layer of calcium-stearate coating applied to lubricated samples makes the defects visually indistinguishable from the normal condition except for Error 3 (peening) in the laboratory trial and Error 4 (scratching) in production trials, an MLP model is able to recognize them almost as accurately as for nonlubricated samples where only the Error 4 (scratching) for the laboratory trial and Error 3 (peening) in production trials are visually indistinguishable from the normal condition.

In Table 2, NC represents normal condition, Err2 represents zinc-phosphate layer only, Err3 represents peening, and Err4 represents scratching; no Err1 representing no coating was introduced, due to possible damage to tooling and sensors in the plant. Confusion matrices here are 4 by 4 matrices, showing how many samples belonging to a particular condition were classified accurately and how many samples were misclassified as other conditions. For example, 10,277 nonlubricated samples were classified accurately as NC, 74 as Err2, 169 as Err3, and 36 as Err4.

Conclusion

This chapter investigates the application of neural networks to the recognition of lubrication defects typical to an industrial cold-forging process employed by fastener manufacturers. Several types of lubrication errors, typical to production of fasteners, were introduced to a set of sample rods drawn both in the laboratory environment and as part of a standard production. The drawing force was measured, from which a limited set of features were extracted. The neural-network-based model learned from these features is able to recognize all types of lubrication errors to a high accuracy. The overall accuracy of the neural-network model is around 95% in both laboratory and production environments with almost uniform distribution of errors between all four errors and the normal condition.

Acknowledgments

The authors wish to thank John Vella for the preparation of test samples and data acquisition and Scott Savage for the design and development of a test rig.

References

- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wansworth and Brooks.
- De A Leao, V. M., Jones, M. H., & Roylance, B. J. (1996). Condition monitoring of industrial machinery through lubricant analysis, *Tribotest*, 2(4), 317-328.
- Frayman, Y., Ting, K. M., & Wang, L. (1999). A fuzzy neural network for datamining: Dealing with the problem of small disjuncts. *Proceedings of the International Joint Conference on Neural Networks (IJCNN'99)*, 4, 2490-2493.
- Kumar, S. A., Ravindra, H. V., & Srinivasa, Y. G. (1997). In-process tool wear monitoring through time series modeling and pattern recognition. *International Journal of Production Research*, 35, 739-751
- Luo, G. Y., Osypiw, D., & Irle, M. (2003). On-line vibration analysis with fast continuous wavelet algorithm for condition monitoring of bearing. *Journal of Vibration and Control*, 9(8), 931-947
- Mattsson, M., Hellman, E., & Ljungberg, S. A. (2001). Airborne thermography for condition monitoring of a public baths buildings. *Proceedings SPIE*, 4360 (pp. 244-251).
- Oh, S. J., Shin, Y. C., & Furgason, E. S. (1994). Surface roughness evaluation via ultrasonic scanning. *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, 41(6), 1-9.
- Pöyhönen, S., Negrea, M., Arkkio, A., Hyötyniemi, H., & Koivo, H. (2002). Fault diagnostics of an electrical machine with multiple support vector classifiers. *Proceedings of the 17th IEEE International Symposium on Intelligent Control (ISIC'02)* (Vol. 1, pp. 373-378).
- Ripley, B. D. (1996). *Pattern recognition and neural networks*. New York: Cambridge University Press.
- Savage, S. C., Kershaw, D. I., & Hodgson, P. D. (1999). Lubrication coating evaluation technique for a cold heading operation. *Proceedings of the IMEA Tooling 99 Conference* (pp. 55-60).
- Weiss, G. M., & Hirsh, H. A. (2000). Quantitative study of small disjuncts. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)* (pp. 665-670).
- Zheng, G. T., & McFadden, P. D. (1999). A time-frequency distribution for analysis of signals with transient components and its application to vibration analysis. *ASME Transactions Journal of Vibration and Acoustics*, 121(3), 328-333.

About the Authors

Rezaul K. Begg received a BSc and MSc in electrical and electronic engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, and a PhD in biomedical engineering from the University of Aberdeen, UK. Currently, he is a faculty member at Victoria University, Melbourne, Australia. Previously, he worked with Deakin University and BUET. He researches in biomedical engineering, biomechanics, and machine learning and has published over 100 research papers in these areas. He is a regular reviewer for several international journals and was on the TPC for a number of major international conferences. He received several awards, including the BUET gold medal and the Chancellor prize for academic excellence.

Joarder Kamruzzaman received a BSc and an MSc in electrical engineering from Bangladesh University of Engineering & Technology, Dhaka, Bangladesh (1986 and 1989, respectively), and a PhD in information system engineering from Muroran Institute of Technology, Japan (1993). Currently, he is a faculty member in the Faculty of Information Technology, Monash University, Australia. His research interest includes computational intelligence, computer networks, bioinformatics, and so on. He has published more than 90 refereed papers in international journals and conference proceedings. He is currently serving as a program committee member of a number of international conferences.

Ruhul A. Sarker obtained his PhD from DalTech (former TUNS), Dalhousie University, Halifax, Canada. He is currently a senior academic at the School of Information Technology and Electrical Engineering, University of New South Wales (UNSW), Canberra, Australia. Before joining UNSW, Dr. Sarker worked with Monash University and

Bangladesh University of Engineering and Technology. He has published over 100 refereed technical papers in international journals, edited reference books, and conference proceedings. He has written two books, edited six reference books and several proceedings, and served as guest editor and technical reviewer for a number of international journals. Dr. Sarker was a technical co-chair of IEEE-CEC 2003 and served many international conferences in the capacity of chair, co-chair, or PC member.

* * *

Hussein A. Abbass is a senior lecturer and the director of the Artificial Life and Adaptive Robotics Laboratory at the School of Information Technology and Electrical Engineering at the Australian Defence Force Academy campus of the University of New South Wales. Dr. Abbass is a senior member of the IEEE and has more than 15 years experience in industry and academia and more than 100 fully refereed papers in international journals and conferences. He teaches computational intelligence related subjects and his research focuses on multiagent systems, data mining, and artificial-life models with applications to defence, security, and business.

Ajith Abraham currently works as a distinguished visiting professor under the South Korean government's Institute of Information Technology Assessment (IITA) professorship programme at Chung-Ang University, Korea. His primary research interests are in computational intelligence with a focus on using evolutionary computation techniques for designing intelligent paradigms. Application areas include several real-world knowledge-mining applications like information security, bioinformatics, Web intelligence, energy management, financial modelling, weather analysis, fault monitoring, multicriteria decision-making and so on. He has associated with over 150 research publications in peer-reviewed reputed journals, book chapters, and conference proceedings of which three have won "best paper" awards. He is the founding coeditor-in-chief of *The International Journal of Hybrid Intelligent Systems* (IJHIS), IOS Press, Netherlands, and the *Journal of Information Assurance and Security* (JIAS). He is also the associate editor of the *International Journal of Systems Science* (IJSS), Taylor & Francis, UK, and *Neurocomputing Journal*, Elsevier Science, The Netherlands. He is also an editorial board member of *Journal of Universal Computer Science* (J.UCS), Springer, Austria; *Journal of Information and Knowledge Management* (JIKM), World Scientific, Singapore; *Journal of Digital and Information Management* (JDIM), Digital Information Research Foundation; and *International Journal of Neural Parallel and Scientific Computations* (NPSC), Dynamic Publishers, Inc., USA. Since 2001, he has been actively involved in the Hybrid Intelligent Systems (HIS) and the Intelligent Systems Design and Applications (ISDA) series of annual international conferences. He is also the general co-chair of the Fourth IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology (WSTST05), Muroran, Japan, and the program co-chair of the Inaugural IEEE Conference on Next Generation Web Services Practices, Seoul, Korea. During the last four years, he has also served the technical committee of over 40 artificial-intelligence-related international conferences and has also given a number of conference tutorials in Europe and USA. He is a member of IEEE, IEEE

(CS), ACM, IEE (UK), IEAust, and also works closely with several academic working groups like EvoNet, EUSFLAT, WFSC, and so on. He received a PhD degree from Monash University, Australia.

Sumit Kumar Bose obtained his bachelor's degree in engineering in mechanical engineering from the University of Delhi (2000). He has approximately one year of experience in automotive design and manufacturing. Presently, he is pursuing a doctoral degree in management information systems at the Indian Institute of Management, Calcutta, India. Sumit's research interest includes production management, operations research, business intelligence, and financial applications of neural networks.

Sergio Cavalieri is an associate professor with the Department of Industrial Engineering of the University of Bergamo. He graduated with a degree in 1994 in management and production engineering. In 1998, he got a PhD in management engineering at the University of Padua. His main fields of interest are modelling and simulation of manufacturing systems, application of multiagent systems and soft-computing techniques (e.g., genetic algorithms, ANNs, expert systems) for operations and supply-chain management. He has been participating in various research projects at the national and international level. He has published two books and about 40 papers in national and international journals and conference proceedings. He is currently coordinator of the IMS Network of Excellence Special Interest Group on Benchmarking of Production Scheduling Systems and a member of the IFAC-TC on Advanced Manufacturing Technology.

Yuehui Chen was born in 1964. He received his BSc in mathematics/automatics from the Shandong University of China (1985), and a PhD in electrical engineering from the Kumamoto University of Japan (2001). During 2001-2003, he had worked as senior researcher of the Memory-Tech Corporation at Tokyo. Since 2003, he has been a member of the faculty of electrical engineering in Jinan University, where he is currently head of the Laboratory of Computational Intelligence. His research interests include evolutionary computation, neural networks, fuzzy systems, hybrid computational intelligence, and their applications in time-series prediction, system identification, and intelligent control. He is the author and coauthor of more than 60 papers. Dr. Chen is a member of IEEE, the IEEE Systems, Man and Cybernetics Society and the Computational Intelligence Society.

Manolis A. Christodoulou was born in Kifissia, Greece, in 1955. He received a diploma from the National Technical University of Athens, Greece, an MS from the University of Maryland, College Park, MD, a degree of engineering from the University of Southern California, Los Angeles, and a PhD from the Democritus University, Thrace, Greece. After being at the University of Patras, he joined the Technical University of Crete, Greece in 1988, where he is currently a professor of control. He has been visiting Georgia Tech, Syracuse University, USC, TUFTS, Victoria University, and MIT. He has authored and co-authored over 160 journal articles, book chapters, books, and conference publications in the areas of control theory and applications, robotics, robotics in medicine, factory

automation, computer integrated manufacturing in engineering, neural networks for dynamic system identification and control, the use of adaptive neural networks for collision avoidance in free flight as well as in scheduling of land-aircraft operations in congested hub airports, and recently systems biology. His works have been cited in international literature by several hundred authors worldwide. He is the organizer of various conferences and sessions of IEEE and IFAC and guest editor in various special issues of international journals. He is managing and cooperating on various research projects in Greece, in the European Union, and in collaboration with the United States. He has held many administrative positions, such as the vice presidency of the Technical University of Crete, chairman of the Office of Sponsored research, and a member of the founding board of governors at the University of Peloponnese. He is also the president of the Automation and Systems European Technology Institute. Dr. Christodoulou is a member of the Technical Chamber of Greece. He has been active in the IEEE CS society as the founder and first chairman of the IEEE CS Greek Chapter. He received the 1997 Best Chapter of the Year Award. He is also the founder of the IEEE Mediterranean Conference on Control and Automation, which has become a successful annual event.

David Enke is an assistant professor of engineering management and systems engineering at the University of Missouri, Rolla, and director of the Laboratory for Investment and Financial Engineering. His research involves the development of intelligent systems, specifically using neural networks and knowledge-based systems in the areas of financial engineering, investment, financial forecasting, capital planning and budgeting, electrical load and price forecasting, and artificial vision.

Yakov Frayman received an MSc from the State Polytechnic Academy of Belarus, ME, from Victoria University of Technology, Australia, and a PhD from Deakin University, Australia. After working in industry for over 20 years as a control/system engineer, he has joined Deakin University in 2000 as a research academic where he is conducting research in neural networks, machine learning, data mining, and machine vision. Dr. Frayman has published over 30 refereed papers in the areas of intelligent modeling and control and the applications of soft computing to manufacturing.

John Fulcher is currently a professor of information technology in the School of IT & Computer Science and director of the Health Informatics Research Centre at the University of Wollongong, Australia. He holds a BEE (Hon.) from the University of Queensland (1972), a research master's degree from LaTrobe University, Melbourne (1981), and a PhD from the University of Wollongong (1999). John is a member of the Association for Computing Machinery and a senior member of the Institute of Electrical & Electronic Engineers. Apart from serving as a preferred reviewer for *ACM Computing Reviews*, he is on the editorial boards of both *Computer Science Education* and the *International Journal of Intelligent & Fuzzy Systems* and serves as reviewer for 13 other journals (including *IEEE Transactions*). His 100 or so publications include a best-selling textbook on microcomputer interfacing, a recent Springer research monograph on applied intelligent systems (coedited with Professor Lakhmi Jain), and three book chapters in the *Oxford University Press Handbook of Neural Computing*. Dr. Fulcher was an invited

keynote speaker at the 5th National Thai Computer Science & Engineering Conference. His research interests include microcomputer interfacing, computer science education, artificial neural networks (especially higher-order ANNs), health informatics, and parallel computing.

Eldon Gunn is professor of industrial engineering at Dalhousie University. He holds a PhD in industrial engineering from the University of Toronto. He served as president of the Canadian Operational Research Society in 1992-1993. From 1996 to 2004, he was head of the Industrial Engineering Department at Dalhousie. He is a registered professional engineer in the Province of Nova Scotia. His research interests include modeling and optimization in both manufacturing and natural-resources settings.

Georgina L. Kelly has a PhD in materials science from Monash University, Australia. She has worked on a number of aspects of mechanical testing and characterisation of materials. She has been at Deakin University for 6 years and leads the surface-performance research team, undertaking both fundamental and contract research. Her current work focuses on surfaces and lubrication, with an emphasis on metal forming.

M. Imad Khan received a BSc in computer science from the University of Karachi where he was associated with the Fuzzy Logic and Soft Computing Research Group for his final thesis. After graduation, Mr. Khan has worked as a software engineer at Avanza Solutions (Pvt) Ltd. In 2002, he has joined Karachi Institute of Information Technology as a research assistant. Currently, he is a doctoral student at the School of Engineering and Technology, Deakin University, Australia.

Mark Kingham has a master's degree in computer science from Edith Cowan University, Perth, Australia. His research interests include adaptive self-learning intelligent systems, and their application to business and financial systems.

Paolo Maccarrone graduated cum laude in management and production engineering (5-year degree) at Politecnico di Milano University in 1992. In 1997, he earned his PhD at the University of Padova, discussing a thesis on "The Post-Audit of Capital Investment Projects." At the moment, he is an associate professor at Politecnico di Milano, where he teaches business economics and organisation and management control systems. He is also member of the faculty of the MIP Politecnico di Milano Business School, where he lectures in management accounting and control systems in the Executive MBA program and other postgraduate master's programs. At MIP he is also co-director of Executive MBA and of the Advanced Course in Information Security Management. His research activities can be grouped into three main fields: (a) management accounting/control and performance measurement systems in for-profit firms (and related executive information systems), (b) corporate social responsibility, (c) the strategic/managerial/organisational aspects of information security management. He is author of several publications on books and international refereed journals. In 2000, he received the

Literati Club Award for the best paper published on the 1999 volume of *Business Process Management Journal*.

Corinne MacDonald received her BEng in industrial engineering from the Technical University of Nova Scotia (1989). She is a registered professional engineer in the Province of Nova Scotia. She is currently a PhD candidate and a lecturer in the Department of Industrial Engineering, Dalhousie University.

Masoud Mohammadian is a member of over 30 international conferences and has chaired several international conferences in computational intelligence and intelligent agents. He has been a keynote speaker at several international conferences on computational intelligence. He is currently a senior lecturer at the school of computing at the University of Canberra in Australia.

Saeid Nahavandi received a BSc (Hon.), MSc and PhD from Durham University, UK. In 1991, he joined Massey University, New Zealand, as a senior lecturer in robotics. In 1998, he became an associate professor at Deakin University, Australia, and the leader for the Intelligent Systems research group. In 2002, Professor Nahavandi took the position of chair in engineering in the same university. Dr. Nahavandi has published over 190 reviewed papers. He is the recipient of four international awards, best paper award at the World Automation Congress, USA, and the Young Engineer of the Year award. Professor Nahavandi is the founder of the World Manufacturing Congress series and the Autonomous Intelligent Systems Congress series. He is a fellow of IEAust and IEE. His current research interests include modeling and control and the application of soft computing to industrial processes.

Stelios E. Perrakis was born in Chania, Crete, Greece, in 1972. He received a BSc in computer science from the University of Crete, Greece (1995) and an MSc in electronic and computer engineering from the Technical University of Crete (2001). His current research interests are in dynamic neural networks and computer implementations for real-time manufacturing-systems scheduling.

Roberto Pinto graduated in management and production engineering (5-year degree) at Politecnico di Milano University in 2001. He is currently assistant professor at the Department of Industrial Engineering of the University of Bergamo. His main research areas are operations and knowledge management in supply chains, business process modelling and simulation, soft-computing techniques like artificial neural networks, genetic algorithms, and constraint programming. He has published about 15 papers on national and international journals and conference proceedings.

Tong-Seng Quah is currently an assistant professor with the School of Electrical and Electronic Engineering, Nanyang Technological University. Dr. Quah lectures in both

undergrad as well as graduate courses such as Software Development Methodology, Software Quality Assurance and Project Management, Object-oriented System Analysis and Design, and Software Engineering. His research interests include financial market modeling using neural networks, software reliability, and e-commerce. Other than academic services, Dr. Quah has undertaken joint projects with major companies in banking and airline industries, as well as statutory boards of the government body. Prior to his academic pursuits, Dr. Quah was a director of a local company dealing with industrial chemicals.

Sadhalaxmi Raipet obtained her master's degree in commerce from the Osmania University (2001). Additionally, she is also a graduate in cost and work accountancy and has completed the company secretary course from the Institute of Company Secretaries of India. Currently, she is in the dissertation phase of the doctoral program in finance and control at the Indian Institute of Management, Calcutta, India. Her research interests include corporate governance, company law, capital markets, regulatory reform, and interest-rate modeling.

Tapabrata Ray obtained his BTech (Hon.), MTech, and PhD from the Indian Institute of Technology (IIT), Kharagpur, India. He worked with three major research institutes in Singapore, namely the Information Technology Institute (ITI), Institute of High Performance Computing (IHPC), and Temasek Laboratories, National University of Singapore. His research interests are in the area of multiobjective and constrained optimization, surrogate-assisted optimization, and robust-design optimization. He is currently a lecturer in the School of Aerospace, Civil and Mechanical Engineering, Australian Defence Force Academy, University of New South Wales, Canberra, Australia.

Bernard F. Rolfe completed a combined economics and engineering degree with honours in 1995 from the Australian National University (ANU). He worked for several years as an IT consultant before starting a PhD at the ANU. His doctorate investigated novel methods of inverse modelling for metal-forming processes and was completed in 2002. This research included an IMechE award-winning journal paper. Currently, he is a lecturer at Deakin University. He is part of a \$1.8 million research project between Deakin and Ford investigating the use of advanced high-strength steels in the automotive industry. He has written over 20 refereed publications.

George A. Rovithakis was born in Chania, Crete, Greece in 1967. He received the diploma in electrical engineering from the Aristotle University of Thessaloniki, Greece (1990) and MS and PhD degrees in electronic and computer engineering both from the Technical University of Crete, Greece (1994 and 1995, respectively). After holding a visiting assistant professor position with the Department of Electronic and Computer Engineering, Technical University of Crete (1995-2002), he joined the Aristotle University of Thessaloniki where he is currently an assistant professor with the Department of Electrical and Computer Engineering. His research interests include nonlinear systems, neural-network systems, robust adaptive control, identification control of unknown

systems using neural networks, production control, intelligent control, fault detection, isolation and accommodation in nonlinear dynamical systems, and automated inspection systems. He has authored or coauthored over 80 publications in scientific journals, referred conference proceedings, and book chapters. He has also coauthored the book *Adaptive Control with Recurrent High-Order Neural Networks* (London, Springer-Verlag, 2000). Dr. Rovithakis serves as a reviewer for various journals and conferences and has served as session chairman or cochairman in international conferences. He is a member of the Technical Chamber of Greece and a senior member of the IEEE.

Janardhanan Sethuraman obtained his BE in electronics and communication engineering from University of Madras with distinction (2001). Currently, Sethuraman is a doctoral student with Management Information Systems group, at the Indian Institute of Management, Calcutta, India. He also specializes in finance and control. Additionally, he is a EURO-IFORS fellow, 2004. His areas of interest include telecommunications design and planning, business cases involving ICT deployments, artificial intelligence, and data mining and its applications in finance. He is interested in applying systems concepts to the development sector, especially e-governance. Sethuraman is one of the founding members of EURO working group on “Young People for OR in Developing Countries” (YORC).

Shuxiang Xu has been a lecturer of computing in the School of Computing at the University of Tasmania, Tasmania, Australia, since 1999. He holds a Bachelor’s of Applied Mathematics (1986) from the University of Electronic Science and Technology of China, Chengdu, China, a Master’s of Applied Mathematics from Sichuan Normal University (1989), Chengdu, China, and a PhD in computing from the University of Western Sydney (2000), Australia. He received an Overseas Postgraduate Research Award from the Australian government in 1996. His current interests include the theory and applications of artificial neural networks and genetic algorithms.

Ming Zhang was born in Shanghai, China. He received an MS in information processing and a PhD in the research area of computer vision from East China Normal University, Shanghai, China (1982 and 1989, respectively). He held postdoctoral fellowships in artificial neural networks with the Chinese Academy of the Sciences in 1989 and the USA National Research Council in 1991. He was a face-recognition airport-security-system project manager and Ph.D. cosupervisor at the University of Wollongong, Australia, in 1992. Since 1994, he has been a lecturer at Monash University, Australia, with a research area of artificial-neural-network financial-information systems. From 1995 to 1999, he was a senior lecturer and PhD supervisor at the University of Western Sydney, Australia, with the research interest of artificial neural networks. He also held a senior research associate fellowship in artificial neural networks with the USA National Research Council in 1999. He is currently an associate professor and graduate-student supervisor in computer science at the Christopher Newport University, VA, USA. With more than 100 papers published, his current research includes artificial neural network models for face recognition, weather forecasting, financial-data simulation, and management.

Index

A

activation function 4
actual costs 201
affine models 125
ANN model for stock selection 154
ARIMA 142
artificial neural network 2, 142, 154, 200, 222

B

backpropagation algorithm 6
bankruptcy prediction 14
Bayesian Regularization algorithm 8, 144
biomedical 2
biscuit sizes 189
Black-Scholes 126
buffers 240

C

cell topology 247
cold forging 263
committed costs 201
condition monitoring 2, 17
connection weights 29
connections 29

control 238
-regulation problem 238
conventional optimization 31
corporate bankruptcy prediction 2
correct
down trend 145
up trend 145
cost estimation 18
credit
ratings 81
scoring 15
crossover 69

D

data-mining 45
deterministic job shop scheduling 237
differential evolution 32
algorithm 33
directional symmetry 145
discretization 238
dispatching times 244
document analysis 2
dynamic
asset-pricing 125
model 240

E

economic factors 156
 engineering tasks 2
 equilibrium models 125
 equity 152
 evolutionary
 algorithm (EA) 29, 222
 multiobjective algorithms 28
 evolving ANNs 28
 exchange rates 81

F

factor loadings 129
 fault diagnosis 2, 18
 feed-forward 124
 ANNs 29
 artificial neural network 64
 neural networks 165
 financial 152
 forecasting 48
 modeling 2
 systems 109
 time-series 81
 flexible neural tree 67
 Fonseca and Fleming's evolutionary
 forecasting 110, 142, 154
 foreign currency exchange
 (forex) 139
 rate 15
 forecasting 2
 forward-rate 127
 fraud detection 2
 fundamental analysis 141
 future-price estimation 2

G

gas-related porosity 187
 generalization 152
 generalized regression neural network 13,
 49, 144
 genetic algorithms 110
 growth factors 156

H

Hajela and Lin's genetic algorithm 31

hidden

 layers 29
 nodes 29
 hierarchical neural network 109
 high-pressure die casting 182
 higher-order ANNs 80
 HONN groups 80
 HONNs 80
 hybrid-learning 67

I

image processing 2
 individual neuron 3
 information-gain 43
 intelligent systems 64
 interest-rate 109, 124
 investment portfolio 15

J

job shops 239
 just-in-time (JIT) production 237

K

kanbans 165

L

Lagrangian relaxation 238
 laguerre function 128
 learning algorithms 5
 life-cycle-costing 201
 linear
 programming 238
 regression 43
 Tanh Axon 130
 liquidity factors 157
 local linear wavelet neural network 64

M

machines 239
 manufacturing 2
 cell formation 2, 20
 control 175
 operations 165
 process 263
 resource planning 237

mean
 absolute error 145
 -squared error 49
 mechanical workshop 247
 modelling 109
 moving window 159
 multilayer perceptron 124
 multimodal 30
 multiobjective optimization 28, 31
 mutation 69

N

Nasdaq 64
 Nelson-Siegel yield curve 124
 network architecture 4, 29
 neural network 70, 109, 238, 264
 -assisted framework 223
 neuroadaptive scheduling methodology 238
 neurons 29
 new evolutionary multiobjective algorithms 31
 niched pareto genetic algorithm 31
 NIFTY 64
 non-differentiable 30
 non-dominated solutions 33
 non-dominated sorting genetic algorithms 31
 nonlinear
 autoregressive models 142
 regression 49
 nonparametric 125
 normalized mean-square error 145
 NP-hard 237

O

operational
 control 166
 decision-support 166
 design 166
 optimization 30
 optimized production timetables 237

P

parameter

optimization 70
 selection 2, 19
 parametric 125
 pareto
 archived evolution strategy 31
 -based differential evolution 32
 -optimal solutions 31
 part types 239
 particle-swarm-optimization 66
 pattern recognition 2
 political factors 156
 polynomial
 coefficients 80
 HONNs 87
 porosity
 formation 184
 models 188
 prediction 81, 109
 present values 127
 probabilistic neural network 11, 51
 processing elements 29
 product costs 199
 production scheduling 2, 19

Q

quality control 2, 20

R

radial basis function 144
 neural network 10
 recognition of lubrication defects 264
 recurrent high-order neural networks 241
 regulation 238
 representation 34
 risk factors 157

S

scaled conjugate gradient 144
 algorithm 7
 scheduling 237
 selection 69
 sheet metal forming 223
 simulation
 metamodels 167
 models 166

- simultaneous evolution 28
- soft computing 64
- splines 125
- standard backpropation 144
- stock
 - market 64
 - analysis 16
 - returns 44
 - performance 2
 - selection 152
- strength pareto evolutionary algorithm 31
- submachines 239
- surrogate models 222

T

- technical analysis 141
- term structure 124
- time series 110
 - analysis 142
- topological structure 29
- tracking 238
- trading and forecasting 2
- transfer function 29
- trigonometric HONNs 80

V

- vector evaluated genetic algorithm 31
- visual interactive simulation (VIS) 168

W

- wavelet neural network 64
- weights 29

Y

- yield
 - factors 156
 - to maturity 127