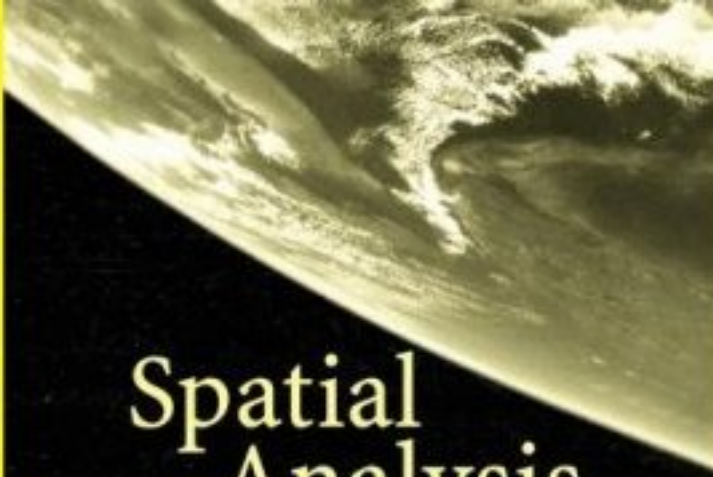


MANFRED M. FISCHER

Spatial Analysis and GeoComputation

Selected Essays

Springer

Spatial Analysis and GeoComputation

Manfred M. Fischer

Spatial Analysis and GeoComputation

Selected Essays

With 53 Figures and 40 Tables

Springer

Prof. Dr. Manfred M. Fischer
Vienna University of Economics and Business Administration
Institute for Economic Geography and GIScience
Nordbergstraße 15/4/A
1090 Vienna, Austria

ISBN-10 3-540-35729-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-35729-2 Springer Berlin Heidelberg New York

Cataloging-in-Publication Data
Library of Congress Control Number: 2006929314

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Cover-Design: Erich Kirchner, Heidelberg

SPIN 11779278 88/3100-5 4 3 2 1 0 – Printed on acid-free paper

Preface

The dissemination of digital spatial databases, coupled with the ever wider use of GISystems, is stimulating increasing interest in spatial analysis from outside the spatial sciences. The recognition of the spatial dimension in social science research sometimes yields different and more meaningful results than analysis which ignores it.

The emphasis in this book is on spatial analysis from the perspective of GeoComputation. GeoComputation is a new computational-intensive paradigm that increasingly illustrates its potential to radically change current research practice in spatial analysis. This volume contains selected essays of Manfred M. Fischer. By drawing together a number of related papers, previously scattered in space and time, the collection aims to provide important insights into novel styles to perform spatial modelling and analysis tasks. Based on the latest developments in estimation theory, model selection and testing this volume develops neural networks into advanced tools for non-parametric modelling and spatial interaction modelling.

Spatial Analysis and GeoComputation is essentially a multi-product undertaking, in the sense that most of the contributions are multi-authored publications. All these co-authors deserve the full credit for this volume, as they have been the scientific source of the research contributions included in the present volume. This book is being published simultaneously with *Innovation, Networks and Knowledge Spillovers*: *Selected Essays.*

I would also like to thank Gudrun Decker, Thomas Seyffertitz and Petra Staufer-Steinnocher for their capable assistance in co-ordinating the various stages of the preparation of the book.

Manfred M. Fischer Vienna, May 2006

Contents

1 Introduction

Traditionally, **spatial analysis** is the domain of the academic discipline of geography, especially of quantitative geography, although ecology, transportation, urban studies and a host of other disciplines draw from and are instrumental in the development of this field (Longley and Batty 1996). Spatial analysis is clearly not a simple and straightforward extension of non-spatial analysis, but raises many distinct problems: the modifiable areal unit problem that consists of two related parts, the scale problem and the zoning problem (see Openshaw 1977); the spatial association problem since the association between spatial units affects the interpretation of georeferenced variables; the spatial heterogeneity problem, and the boundary effects problem. By taking these problems into account, the spatial analyst gives more meaning to the subject. The value of spatial analysis comes from its ability to yield insights about phenomena and processes that occur in the real world.

Spatial analysis, as it evolved over the past few decades, consists of two major areas of research: spatial data analysis [in a more strict sense] and spatial modelling though the boundary is rather blurred (see Fischer and Getis 1997). Spatial modelling lies at the heartland of regional science and includes a wide range of different models (see Wegener and Fotheringham 2000), most notably models of location-allocation (see, for example, Church and Revelle 1976), spatial interaction (see, for example, Sen and Smith 1975, Roy 2004, Fischer and Reggiani 2004), and spatial choice and search (see, for example, Ben-Akiva and Lerman 1985, Fischer et al. 1990, Fischer and Nijkamp 1985, 1987) and spatial dynamic analysis (see, for example, Donaghy 2001, Nijkamp and Reggiani 1998). Spatial data analysis includes procedures for the identification of the characteristics of georeferenced data, tests on hypotheses about patterns and relationships, and construction of models that give meaning to patterns and relationships among georeferenced variables.

The breadth of interest in spatial data analysis is evident from earlier books and edited volumes in the field: Ripley (1981), Upton and Fingleton (1985), Anselin (1988), Griffith (1988), Haining (1990), Cressie (1991), Fischer and Nijkamp (1993), Fotheringham and Rogerson (1994), Bailey and Gatrell (1995), Fischer et al. (1996), and Longley and Batty (1996). The continued vitality of the field over the last decade is illustrated by the increasing recognition of the spatial dimension in social science research that sometimes yields different and more meaningful results than analysis that ignores it. The expanding use of spatial analysis methods and techniques reflects the significance of location and spatial interaction in

theoretical frameworks, most notably in the new economic geography as embodied in the work of Krugman (1991a, 1991b), Fujita et al. (1999) and others. Central to the new economic geography is an explicit accounting for location and spatial interaction in theories of trade and economic development. The resulting models of increasing returns and imperfect competition yield various forms of spatial externalities and spillovers whose spatial manifestation requires a spatial analytic approach in empirical work (Goodchild et al. 2000).

The technology of spatial analysis has been greatly affected by computers. In fact, the increasing interest in spatial analysis in recent years is directly associated with the ability of computers to process large amounts of spatial data and to map data very quickly and cheaply. Specialised software for the capture, manipulation and presentation of spatial data, which can be referred to as Geographical Information Systems [GIS], has widely increased the range of possibilities of organising spatial data by new and efficient ways of spatial integration and spatial interpolation. Coupled with the improvements in data availability and increases in computer memory and speed, these novel techniques open up new ways of working with geographic information. Spatial analysis is currently entering a period of rapid change characterised by **GeoComputation**, a new large-scale and computationally intensive scientific paradigm (see Longley et al. 1998, Openshaw and Abrahart 2000, Openshaw et al. 2000, Fischer and Leung 2001).

The principal driving forces behind this paradigm are four-fold: First, the increasing complexity of spatial systems whose analysis requires new methods for modelling nonlinearities, uncertainty, discontinuity, self-organisation and continual adaptation; second, the need to find new ways of handling and utilising the increasingly large amounts of spatial information from the geographic information systems [GIS] and remote sensing [RS] data revolutions; third, the increasing availability of computational intelligence [CI] techniques that are readily applicable to many areas in spatial analysis; and fourth, developments in high performance computing that are stimulating the adoption of a computational paradigm for problem solving, data analysis and modelling. But it is important to note that not all GeoComputation based research needs the use of very large data sets or requires access to high performance computing.

The present collection of papers is intended as a convenient resource, not only for the results themselves, but also for the concepts, methods and techniques useful in obtaining new results or extending results presented here. The articles of this volume may thus serve usefully as supplemental readings for graduate students and senior researchers in spatial analysis from the perspective of GeoComputation. We have chosen articles and book chapters which we feel should be made accessible not only to specialists but to a wider audience as well. By bringing together this specific selection of articles and book chapters and by presenting them as a whole, this collection is a novel combination.

The book is structured into four parts. PART I sets the context by dealing with broader issues connected with GIS and spatial analysis. The chapters included have been written for more general audiences. Spatial analysis is reviewed as a technology for analysing spatially referenced data and GIS as a technology comprising a set of computer-based tools designed to store, process, manipulate,

explore, analyse, and present spatially identified information. PART II deals with key computational intelligence technologies such as neural networks and evolutionary computation. Much of the recent interest in these technologies stems from the growing realisation of the limitations of conventional statistical tools and models as vehicles for exploring patterns and relationships in data-rich environments and from the consequent hope that these limitations may be overcome by the judicious use of neural net approaches and evolutionary computation. These technologies promise a new style of performing spatial modelling and analysis tasks in geography and other spatial sciences. This new style gives rise to novel types of models, methods and techniques which exhibit various aspects of computational intelligence. The focus of PART III is on neural pattern classification in remote sensing environments. It provides the necessary theoretical framework, reviews many of the most important algorithms for optimising the values of parameters in a network and – through various examples – displays the efficient use of adaptive pattern classifiers as implemented with the fuzzy ARTMAP system and with error-based learning systems based upon single hidden layer feedforward networks. Anyone interested in recent advances in neural spatial interaction modelling may wish to look at the final part of the volume which covers the latest, most significant developments in estimation theory, and provides a number of insights into the problem of generalisation.

PART I Spatial Analysis and GIS

PART I of the present volume is composed of four contributions:

- Spatial Analysis in Geography (Chapter 2)
- Spatial Interaction Models and the Role of Geographic Information Systems (Chapter 3),
- GIS and Network Analysis (Chapter 4), and
- Expert Systems and Artificial Neural Networks for Spatial Analysis and Modelling (Chapter 5).

These four contributions largely drawing on the work done in the GISDATA research network of the European Science Foundation [1993-1997] will now be briefly discussed.

Chapter 2, a state-of-the-art review of spatial analysis that has found entry in Elsevier's *International Encyclopedia of the Social and Behavioral Sciences*, views spatial analysis as a technology for analysing spatially referenced object data, where the objects are either points [spatial point patterns, i.e. point locations at which events of interest have occurred] or areas [area or lattice data, defined as discrete variations of attributes over space]. The need for spatial analytic techniques relies on the widely shared view that spatial data are special and require a specific type of data processing. Two unique properties of spatial data are worthwhile to note: spatial dependency and spatial heterogeneity. Spatial depen-

dency is the tendency for things closer in geographic space to be more related while spatial heterogeneity is the tendency of each location in geographic space to show some degree of uniqueness. These features imply that systems and tools to support spatial data processing and decision making must be tailored to recognise and exploit the unique nature of spatial data.

The review charts the considerable progress that has been made in developing advanced techniques for both exploratory and model driven spatial data analysis. Exploratory spatial data analysis [ESDA], not widely used until the late 1980s, includes among other activities the identification of data properties and the formulation of hypotheses from data. It provides a methodology for drawing out useful information from data. Model driven analysis of spatial data relies on testing hypotheses about patterns and relationships, utilising a range of techniques and methodologies for hypothesis testing, the determination of confidence intervals, estimation of spatial models, simulation, prediction, and assessment of model fit.

The next chapter views GIS as context for spatial analysis and modelling. GIS is a powerful application-led technology that comprises a set of computer-based tools designed to store, process, manipulate, explore, analyse and present geographic information. Geographic Information [GI] is defined as information referenced to specific locations on the surface of the Earth. Time is optional, but location is essential and the element that distinguishes GI from all other types of information. Locations are the basis for many of the benefits of GISystems: the ability to visualise in form of maps, the ability to link different kinds of information together because they refer to the same location, or the ability to measure distances and areas. Without locations, data have little value within a GISystem (Longley et al. 2001). The functional complexity of GISystems is what it makes it different from other information systems.

Many of the more sophisticated techniques and algorithms to process spatial data in spatial models are currently, however, not or hardly available in conventional GISystems. This raises the question of how spatial models may be integrated with GISystems. Nyerges (1992) suggested a conceptual framework for the coupling of spatial analysis routines with GISystems that distinguishes four categories with increasing intensity of coupling: first, isolated applications where the GIS and the spatial analysis programme are run in different hardware environments and data transfer between the possibly different data models is performed by ASCII files off-line; second, loose coupling where coupling by means of ASCII or binary files is carried out online on the same computer or different computers in a network; third, tight coupling through a standardised interface without user intervention; and fourth, full integration where data exchange is based on a common data model and database management system.

Chapter 3 discusses possibilities and problems of interfacing spatial interaction models and GISystems from a conceptual rather than a technical point of view. The contribution illustrates the view that the integration between spatial analysis/ modelling and GIS opens up tremendous opportunities for the development of new, highly visual, interactive and computational techniques for the analysis of spatial data that are associated with a link or pair of locations [points, areas] in geographic space. Using the Spatial Interaction Modelling [SIM] software

package, developed at the Institute for Economic Geography and GIScience, as an example, the chapter suggests that in spatial interaction modelling GIS functionalities are especially useful in three steps of the modelling process: zone design, matrix building and visualisation.

The next chapter [**Chapter 4**], written for the *Handbook of Transport Geography and Spatial Systems* [edited by D.A. Hensher, K. J. Button, K. E. Haynes and P. R. Stopher], moves attention to GIS-T, the application of GISystems to research, planning and management in transportation. While the strengths of standard GIS technology are in mapping display and geodata processing, GIS-T requires new data structures to represent the complexities of transportation networks and to perform different network algorithms in order to fulfil its potential in the field of logistics and distribution logistics.

The chapter discusses data model and design issues that are specifically oriented to GIS-T, and identifies several improvements of the traditional network data model that are required to support advanced network analysis in a ground transportation context. These improvements include turn-tables, dynamic segmentation, linear referencing, traffic lines and non-planar networks. Most commercial GISystems software vendors have extended their basic GIS data model during the past two decades to incorporate these innovations (Goodchild 1998). The paper shifts attention also to network routing problems that have become prominent in GIS-T: the traveling-salesman problem, the vehicle-routing problem and the shortest-path problem with time windows, a problem that occurs as a subproblem in many time-constrained routing and scheduling issues of practical importance. Such problems are conceptually simple, but mathematically complex and challenging. The focus is on theory and algorithms for solving these problems.

Present-day GISystems are – in essence – geographic database management systems with powerful visualisation capabilities. To provide better support for spatial decision making in a GISystem should contain not only information, but knowledge and should, moreover, possess common-sense and technical reasoning capabilities. Therefore, it is essential to require a GISystem to have the following additional capabilities in the context of spatial decision support (Leung 1997): first, a formalism for representing loosely structured spatial knowledge; second, a mechanism for making inference with domain specific knowledge and for making common sense reasoning; third, facilities to automatically acquire knowledge or to learn by examples; and finally, intelligent control over the utilisation of spatial information, declarative and procedural knowledge. This calls for the integrative utilisation of state-of-the-art procedures in artificial and computational intelligence, knowledge engineering, software engineering, spatial information processing and spatial decision theory.

The final contribution to PART I, **Chapter 5**, outlines the architecture of a knowledge based GISystem that has the potential of supporting decision making in a GIS environment, in a more intelligent manner. The efficient and effective integration of spatial data, spatial analytic procedures and models, procedural and declarative knowledge is through fuzzy logic, expert system and neural network technologies. A specific focus of the discussion is on the expert system and neural

network components of the system, technologies which had been relatively unknown in the GIS community at the time this chapter was written.

PART II Computational Intelligence in Spatial Data Analysis

Novel modes of computation which are collectively known as **C**omputational **I**ntelligence [CI]-technologies hold some promise to meet the needs of spatial data analysis in data-rich environments (see Openshaw and Fischer 1995). Computational intelligence refers to the lowest level forms of intelligence stemming from the ability to process numerical data, without explicitly using knowledge in an artificial intelligence sense. The raison d'être of CI-based modelling is to exploit the tolerance for imprecision and uncertainty in large-scale spatial problems, with an approach characterised by robustness and computational adaptivity (see Fischer and Getis 1997). Evolutionary computation including genetic algorithms, evolution strategies and evolutionary programming; and neural networks also known as neurocomputing are the major representative components in this arena. Three contributions have been chosen for Part II. These are as follows:

- Computational Neural Networks – Tools for Spatial Data Analysis (Chapter 6),
- Artificial Neural Networks: A New Approach to Modelling Interregional Telecommunication Flows (Chapter 7), and
- A Genetic-Algorithms Based Evolutionary Computational Neural Network for Modelling Spatial Interaction Data (Chapter 8).

Chapter 6 is essentially a tutorial text that gives an introductory exposure to computational neural networks for students and professional researchers in spatial data analysis. The text covers a wide range of topics including a definition of computational neural networks in mathematical terms, and a careful and detailed description of computational neural networks in terms of the properties of the processing elements, the network topology and learning in the network. The chapter presents four important families of neural networks that are especially attractive for solving real world spatial analysis problems: backpropagation networks, radial basis function networks, supervised and unsupervised ART models, and self-organising feature map networks. With models of the first three families we will be working in the chapters that follow.

In contrast to Chapter 6 the two other chapters in PART II represent pioneering contributions. **Chapter 7**, written with Sucharita Gopal [Boston University], represents a clear break with traditional methods for explicating spatial interaction. The paper presented at the 1992 Symposium of the IGU-Commission on Mathematical Models at Princeton University opened up the development of a novel style for geocomputational models and techniques in spatial data analysis that exhibits various facets of computational intelligence. The paper presents a new

approach for modelling interactions over geographic space, one which has been a clear break with traditional methods used so far for explicating spatial interaction.

The approach suggested is based upon a general nested sigmoid neural network model. Its feasibility is illustrated in the context of modelling interregional telecommunication traffic in Austria and its performance evaluated in comparison with the classical regression approach of the gravity type. The application of this neural network may be viewed as a three-stage process. The *first stage* refers to the identification of an appropriate model specification from a family of single hidden layer feedforward networks characterised by specific nonlinear hidden processing elements, one sigmoidal output and three input elements. The input-output dimensions had been chosen in order to make the comparison with the classical gravity model as close as possible. The *second stage* involves the estimation of the network parameters of the chosen neural network model. This is performed by means of combining the sum-of-squares error function with the error back-propagating technique, an efficient recursive procedure using gradient descent information to minimise the error function. Particular emphasis is laid on the sensitivity of the network performance to the choice of initial network parameters as well as on the problem of overfitting. The *final stage* of applying the neural network approach refers to testing and evaluating the out-of-sample [generalisation] performance of the model. Prediction quality is analysed by means of two performance measures, average relative variance and the coefficient of determination, as well as by the use of residual analysis.

In a sense, the next chapter [**Chapter 8**], written with Yee Leung [Chinese University of Hongkong], takes up where Chapter 7 left off, the issue of determining a problem adequate network topology. With the view of modelling interactions over geographic space, Chapter 8 considers this problem as a global optimisation problem and proposes a novel approach that embeds backpropagation learning into the evolutionary paradigm of genetic algorithms. This is accomplished by interweaving a genetic search for finding an optimal neural network topology with gradient-based backpropagation learning for determining the network parameters. Thus, the model builder will be released from the burden of identifying appropriate neural network topologies that will allow a problem to be solved with simple, but powerful learning mechanisms, such as backpropagation of gradient descent errors. The approach is applied to the family of three inputs, single hidden layer, single output feedforward models using interregional telecommunication traffic data for Austria to illustrate its performance and to evaluate its robustness.

PART III GeoComputation in Remote Sensing Environments

There is a long tradition on spatial pattern recognition that deals with classifications utilising pixel-by-pixel spectral information from satellite imagery. Classification of terrain cover from satellite imagery represents an area of considerable interest and research today. Satellite sensors record data in a variety of spectral

channels and at a variety of ground resolutions. The analysis of remotely sensed data is usually achieved by machine-oriented pattern recognition of which classification based on maximum likelihood, assuming Gaussian distribution of the data, is the most widely used one. Research on neural pattern classification started around 1990. The first studies established the feasibility of error-based learning systems such as backpropagation networks (see, for example, McClellan et al. 1989, Benediktsson et al. 1990). Subsequent studies analysed backpropagation networks in some more detail and compared them to standard statistical classifiers such as the Gaussian maximum likelihood.

The focus of PART III is on adaptive spectral pattern classifiers as implemented with backpropagation networks, radial basis function networks and fuzzy ARTMAP. The following three papers have been chosen for this part of the book:

- Evaluation of Neural Pattern Classifiers for a Remote Sensing Application (Chapter 9),
- Optimisation in an Error Backpropagation Neural Network Environment with a Performance Test on a Spectral Pattern Classification Problem (Chapter 10), and
- Fuzzy ARTMAP – A Neural Classifier for Multispectral Image Classification (Chapter 11).

The spectral pattern recognition problem in these chapters is the supervised pixel-by-pixel classification problem in which the classifier is trained with examples of the classes [categories] to be recognised in the data set. This is achieved by using limited ground survey information which specifies where examples of specific categories are to be found in the imagery. Such ground truth information has been gathered on sites which are well representative of the much larger area analysed from space. The image data set consists of 2,460 pixels [resolution cells] selected from a Landsat-5 Thematic Mapper [TM] scene [270x360 pixels] from the city of Vienna and its northern surroundings [observation date: June 5, 1985; location of the centre: 16°23'E, 48°14'N; TM Quarter scene 190-026/4]. The six Landsat TM spectral bands used are blue [SB1], green [SB2], red [SB3], near IR [SB4], mid IR [SB5] and mid IR [SB7], excluding the thermal band with only a 120 m ground resolution. Thus, each TM pixel represents a ground area of 30x30 m^2 and its six spectral band values ranging over 256 digital numbers [8 bits].

Chapter 9, written with Sucharita Gopal [Boston University], Petra Staufer [Vienna University of Economics and Business Administration] and Klaus Steinnocher [Austrian Research Centers Seibersdorf] represents the research tradition of adaptive spectral pattern recognition and evaluating the generalisation performance of three adaptive classification, the radial basis function network and two backpropagation networks differing in the type of hidden layer specific transfer functions, in comparison to the maximum likelihood classifier. Performance is measured in terms of the map user's, the map producer's and the total classification accuracy. The study demonstrates the superiority of the neural classifiers, but also illustrates that small changes in network design, control

parameters and initial conditions of the backpropagation training process might generate large changes in the behaviour of the classifiers, a problem that is often neglected in neural pattern classification.

The next chapter [**Chapter 10**], written with Petra Staufer [Vienna University of Economics and Business Administration] develops a mathematically rigid framework for minimising the cross-entropy error function – an important alternative to the sum-of-squares error function that is widely used in research practice – in an error backpropagating framework. Various techniques of optimising this error function to train single hidden layer neural classifiers with softmax output transfer functions are investigated on the given real world pixel-by-pixel classification problem. These techniques include epoch-based and batch versions of backpropagation of gradient descent, Polak-Ribière conjugate gradient and Broyden-Fletcher-Goldfarb-Shanno quasi-Newton errors. It is shown that the method of choice depends upon the nature of the learning task and whether one wants to optimise learning for speed or classification performance.

The final chapter in PART III, **Chapter 11**, shifts attention to the Adaptive Resonance Theory of Carpenter and Grossberg (1987a, b), which is closely related to adaptive versions of *k*-means such as ISODATA. Adaptive resonance theory provides a large family of models and algorithms, but limited analysis has been performed of their properties in real world environments. The chapter, written with Sucharita Gopal [Boston University], analyses the capability of the neural pattern recognition system, fuzzy ARTMAP, to generate classifications of urban land cover, using the given remotely sensed image. Fuzzy ARTMAP synthesises fuzzy logic and Adaptive Resonance Theory [ART] by exploiting the formal similarity between the computations of fuzzy subsets and the dynamics of category choice, search and learning. The chapter describes design features, system dynamics and simulation algorithms for this learning system, which is trained and tested for classification [with eight classes a priori given] of the multispectral image of the given Landsat-5 Thematic Mapper scene from the city of Vienna on a pixel-by-pixel basis. The performance of the fuzzy ARTMAP is compared with that of an error-based learning system based upon a single hidden layer feedforward network, and the Gaussian maximum likelihood classifier as conventional statistical benchmark on the same database. Both neural classifiers outperform the conventional classifier in terms of classification accuracy. Fuzzy ARTMAP leads to out-of-sample classification accuracies which are very close to maximum performance, while the backpropagation network – like the conventional classifier – has difficulty in distinguishing between the land use categories.

PART IV New Frontiers in Neural Spatial Interaction Modelling

Spatial interaction models represent a class of methods which are appropriate for modelling data that are associated with a link or pair of locations [points, areas] in geographic space. They are used to describe and predict spatial flows of people,

commodities, capital and information over geographic space. Neural spatial interaction models represent the most recent innovation in the design of spatial interaction models. The following three papers have been chosen to represent new frontiers in neural spatial interaction modelling:

- Neural Network Modelling of Constrained Spatial Interaction Flows: Design, Estimation, and Performance Issues (Chapter 12),
- Learning in Neural Spatial Interaction Models: A Statistical Perspective (Chapter 13), and
- A Methodology for Neural Spatial Interaction Modelling (Chapter 14).

In the recent past, interest has focused largely – not to say exclusively – on unconstrained neural spatial interaction models. These models represent a rich and flexible family of spatial interaction function approximators, but they may be of little practical value if a priori information is available on accounting constraints on the predicted flows. **Chapter 12**, written with Martin Reismann and Katerina Hlavackova-Schindler [both Vienna University of Economics and Business Administration], presents a novel neural network approach for the case of origin- or destination-constrained spatial interaction flows. The proposed approach is based on a modular network design with functionally independent product unit network modules where modularity refers to a decomposition on the computational level. Each module is a feedforward network with two inputs and a hidden layer of product units, and terminates with a single summation unit. The prediction is achieved by combining the outcome of the individual modules using a nonlinear normalised transfer function multiplied with a bias term that implements the accounting constraint. The efficacy of the model approach is demonstrated for the origin-constrained case of spatial interaction using Austrian interregional telecommunication traffic data, in comparison to the standard origin-constrained gravity model.

The chapter that follows, **Chapter 13**, is a convenient resource for those interested in a statistical view of neural spatial interaction modelling. Neural spatial interaction models are viewed as an example of non-parametric estimation that makes few – if any – a priori assumptions about the nature of the data-generating process to approximate the true, but unknown spatial interaction function of interest. The chapter develops a rationale for specifying the maximum likelihood learning problem in product unit neural networks for modelling origin-constrained spatial interaction flows as introduced in the previous chapter. The study continues to consider Alopex based global search, in comparison to local search based upon backpropagation of gradient descents, to solve the maximum likelihood learning problem. An interesting lesson from the results of the study and an interesting avenue for further research is to make global search more speed efficient. This may motivate the development of a hybrid procedure that uses global search to identify regions of the parameter space containing promising local minima and gradient information to actually find them.

In the final chapter [**Chapter 14**], written with Martin Reismann [Vienna University of Economics and Business Administration], an attempt is made to develop

a mathematically rigid and unified framework for neural spatial interaction modelling. Families of classical neural network models, but also less classical ones such as product unit neural network ones are considered for both, the cases of unconstrained and singly constrained spatial interaction flows. Current practice in neural network modelling appears to suffer from least squares and normality assumptions that ignore the true integer nature of the flows and approximate a discrete-valued process by an almost certainly misrepresentative continuous distribution. To overcome this deficiency the study suggests a more suitable estimation approach, maximum likelihood estimation under more realistic distributional assumptions of Poisson processes, and utilises a global search procedure, such as Alopex, to solve the maximum likelihood estimation problem. To identify the transition from underfitting to overfitting the data are split into training, internal validation, and test sets. The bootstrapping pairs approach with replacement is adopted to combine the purity of data splitting with the power of a resampling procedure to overcome the generally neglected issue of fixed data splitting and the problem of scarce data. The approach shows the power to provide a better statistical picture of the prediction variability.

References

Anselin L. (1988): *Spatial Econometrics: Methods and Models*, Kluwer Academic Publishers, Dordrecht, Boston, London

Bailey T.C. and Gatrell A.C. (1995): *Interactive Spatial Data Analysis*, Longman, Essex

Ben-Akiva M. and Lerman S.R. (1985): *Discrete Choice. Theory and Application to Travel Demand*, MIT Press, Cambridge [MA] and London [UK]

Benediktsson J.A., Swain P.H. and Ersoy O.K. (1990): Neural network approaches versus statistical methods in classification of multisource remote sensing data, *IEEE Transactions on Geoscience and Remote Sensing* 28 (4), 540-552

Carpenter G.A. and Grossberg S. (1987a): A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics, and Image Processing* 37, 54-115

Carpenter G.A. and Grossberg S. (1987b): ART 2 stable self-organizing of pattern recognition codes for analog input patterns, *Applied Optics* 26 (3), 4919-4930

Church R.L. and ReVelle C.S. (1976): Theoretical and computational links between the p-median, location set-covering and the maximal covering location problems, *Geographical Analysis* 8 (1), 406-415

Cressie N.A.C. (1991): *Statistics for Spatial Data*, John Wiley, Chichester [UK], New York

Donaghy K.P. (2001): Solution and econometric estimation of spatial dynamic models in continuous space and continuous time, *Journal of Geographical Systems* 3 (3), 257-270

Fischer M.M. (2002): A novel modular product unit neural network for modelling constrained spatial interaction flows. In: *Proceedings of the IEEE 2002 World Congress on Computational Intelligence: 2002 Congress on Evolutionary Computation*, IEEE Press, Piscataway [NJ], pp. 1215-1220

Fischer M.M. (1994): From conventional to knowledge based geographic information systems, *Computers, Environment and Urban Systems* 18 (4), 233-242

Fischer M.M. and Getis A. (1999): New advances in spatial interaction theory, *Papers in Regional Science* 78 (2), 117-118

Fischer M.M. and Getis A. (eds.) (1997): *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg , New York

Fischer M.M. and Leung Y. (2001): GeoComputational modelling: Techniques and applications: Prologue. In: Fischer M.M. and Leung Y. (eds.) *GeoComputational Modelling: Techniques and Applications*, Springer, Berlin, Heidelberg, New York, pp. 1-12

Fischer M.M. and Nijkamp P. (eds.) (1993): *Geographic Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York

Fischer M.M. and Nijkamp P. (1987): From static towards dynamic discrete choice modelling, *Regional Science and Urban Economics* 17 (1), 3-27

Fischer M.M. and Nijkamp P. (1985): Developments in explanatory discrete spatial data and choice analysis, *Progress in Human Geography* 9 (4), 515-551

Fischer M.M. and Reggiani A. (2004): Spatial interaction models: From the gravity to the neural network approach. In: Cappello R. and Nijkamp P. (eds.) *Urban Dynamics and Growth. Advances in Urban Economics*, Elsevier, Amsterdam [= Contributions to Economic Analysis 266], pp. 319-346

Fischer M.M., Nijkamp P. and Papageorgiou Y.Y. (eds.) (1990): *Spatial Choices and Processes*, North-Holland, Amsterdam

Fischer M.M., Scholten H.J. and Unwin D. (eds.) (1996): *Spatial Analytical Perspectives on GIS*, Taylor & Francis, Basingstoke [=GISDATA 4]

Fotheringham A.S. and Rogerson P. (eds.) (1994): *Spatial Analysis and GIS*, Taylor & Francis, London

Fujita M., Krugman P. and Venables A.J. (1999): *The Spatial Economy: Cities, Regions, and International Trade*, MIT Press, Cambridge [MA]

Goodchild M.F. (1998): Geographical information systems and disaggregate transportation modeling, *Geographical Systems* 5 (1), 13-44

Goodchild M.F., Anselin L., Appelbaum R.P. and Harthorn B.H. (2000): Toward spatially integrated social science, International Regional Science Review 23 (2), 139-159

Griffith D.A. (2003): *Spatial Autocorrelation and Spatial Filtering*, Springer, Berlin, Heidelberg, New York

Griffith D.A (1988): *Advanced Spatial Statistics: Special Topics in the Exploration of Quantitative Spatial Data Series*, Kluwer Academic Publishers, Dordrecht, Boston, London

Haining R. (1990): *Spatial Data Analysis in the Social Sciences*, Cambridge University Press, Cambridge

Krugman P. (1991a): *Geography and Trade*, MIT Press, Cambridge [MA]

Krugman P. (1991b): Increasing returns and economic geography, *Journal of Political Economy* 99 (3), 483-499

Leung Y. (1997): *Intelligent Spatial Decision Support Systems*, Springer, Berlin, Heidelberg, New York

Longley P.A. and Batty M. (eds.) (1996): *Spatial Analysis: Modelling in a GIS Environment*. GeoInformatica International, Cambridge

Longley P.A., Brooks S.M., McDonnell R. and Macmillan B. (eds.) (1998): *Geocomputation. A Primer*, John Wiley, Chichester [UK], New York

Longley P.A., Goodchild M.F, Maguire D.J. and Rhind D.W. (eds.) (2001): *Geographical Information Systems and Science*, John Wiley, Chichester [UK], New York

McClellan G.E., DeWitt R.N., Hemmer T.H., Matheson L.N. and Moe G.O. (1989): Multispectral Image-Processing with a Three-Layer Backpropagation Network. In: *Pro-*

ceedings of the 1989 International Joint Conference on Neural Networks, Washington, D.C., pp. 151-153

Nijkamp P. and Reggiani A. (1998): *The Economics of Complex Spatial Systems*, Elsevier, Amsterdam

Nyerges T.L. (1992): Coupling GIS and spatial analytical models. In: Bresnahan P., Corwin E. and Cowan D.J. (eds.) *Proceedings of the Fifth International Symposium on Spatial Data Handling*, University of South Carolina, Columbia [SC], pp. 534-542

Openshaw S. (1977): A geographical solution to scale and aggregation problems in region-building, partitioning, and spatial modelling, *Transactions of the Institute of British Geographers*, *New Series* 2, 459-472

Openshaw S. and Abrahart R.J. (eds.) (2000): *GeoComputation*, Taylor & Francis, London

Openshaw S. and Fischer M.M. (1995): A framework for research on spatial relevant to GEO-statistical information systems, *Geographical Systems* 2 (4), 325-337

Openshaw S., Fischer M.M., Benwall G. and Macmillan B. (2000): GeoComputation research agendas and futures. In: Openshaw S. and Abrahart R.J. (eds.) *GeoComputation*, Taylor & Francis, London, pp. 379-400

Ripley B.D. (1981): *Spatial Statistics*, John Wiley, Chichester [UK], New York

Roy J.R (2004): *Spatial Interaction Modelling. A Regional Context*, Springer, Berlin, Heidelberg, New York

Sen A. and Smith T.E (1995): *Gravity Models of Spatial Interaction Behavior*, Springer, Berlin, Heidelberg, New York

Upton G. and Fingleton B. (1985) *Spatial Data Analysis by Example*, John Wiley, Chichester [UK], New York

Wegener M. and Fotheringham S. (eds.) (2000): *Spatial Models and GIS*: *New Potential and New Models*, Taylor & Francis, London

Yao X., Fischer M.M. and Brown G. (2001): Neural network ensembles and their application to traffic flow prediction in telecommunication networks. In: *Proceedings of the 2001 IEEE-INNS-ENNS International Joint Conference on Neural Networks*, IEEE Press, Piscataway [NJ], pp. 693-698

Part I

Spatial Analysis and GIS

2 Spatial Analysis in Geography

The proliferation and dissemination of digital spatial databases, coupled with the ever wider use of Geographic Information Systems (*GISystems or briefly GIS*), *is stimulating increasing interest in spatial analysis from outside the spatial sciences. The recognition of the spatial dimension in social science research sometimes yields different and more meaningful results than analysis that ignores it. Spatial analysis is a research paradigm that provides a unique set of techniques and methods for analysing events – events in a very general sense – that are located in geographical space* (*see Table 1*). *Spatial analysis involves spatial modelling, which includes models of location-allocation, spatial interaction, spatial choice and search, spatial optimisation, and space-time. This article concentrates on spatial data analysis, the heart of spatial analysis.*

1 Spatial Data and the Tyranny of Data

Spatial data analysis focuses on detecting patterns and exploring and modelling relationships between such patterns in order to understand processes responsible for observed patterns. In this way, spatial data analysis emphasises the role of space as a potentially important explicator of socioeconomic systems, and attempts to enhance understanding of the working and representation of space, spatial patterns, and processes.

1.1 Spatial Data and Data Types

Empirical studies in the spatial sciences routinely employ data for which locational attributes are an important source of information. Such data characteristically consist of one or few cross-sections of observations for either micro-units such as individuals (households, firms) at specific points in space, or aggregate spatial entities such as census tracts, electoral districts, regions, provinces, or even countries. Observations such as these, for which the absolute location and/or relative positioning (spatial arrangement) is explicitly taken into account, are termed *spatial data*.

Table 1 Popular techniques and methods in spatial data analysis

	Exploratory spatial data analysis	*Model driven spatial data analysis*
Object data		
Point pattern	Quadrat methods Kernel density estimation Nearest neighbour methods *K* function analysis	Homogeneous and heterogeneous Poisson process models, and multivariate extensions
Area data	Global measures of spatial associations: Moran's *I*, Geary's *c*	Spatial regression models
	Local measures of spatial association: G_i and G_i^* statistics, Moran's scatter plot	Regression models with spatially autocorrelated residuals
Field data	Variogram and covariogram Kernel density estimation Thiessen polygons	Trend surface models Spatial prediction and kriging Spatial general linear modelling
Spatial interaction data	Exploratory techniques for representing such data	Spatial interaction models
	Techniques to uncover evidence of hierarchical structure in the data such as graph-theoretic and regionalisation techniques	Location-allocation models
		Spatial choice and search models
		Modelling paths and flows through a network

In the socioeconomic realm points, lines, and areal units are the fundamental entities for representing spatial phenomena. This form of spatial referencing is also a salient feature of GISystems. Three broad classes of spatial data can be distinguished:

(a) *object data* where the objects are either *points* [spatial point patterns or locational data, i.e. point locations at which events of interest have occurred] or *areas* [area or lattice data, defined as discrete variations of attributes over space],

(b) *field data* [also termed geostatistical or spatially continuous data], that is, observations associated with a continuous variation over space, given values at fixed sampling points, and

(c) *spatial interaction data* [sometimes called link or flow data] consisting of measurements each of which is associated with a link or pair of locations representing points or areas.

The analysis of spatial interaction data has a long and distinguished history in the study of a wide range of human activities, such as transportation movements, migration, and the transmission of information. Field data play an important role in the environmental sciences, but are less important in the social sciences. This article therefore focuses on object data, the most appropriate perspective for spatial analysis applications in the social sciences. Object data include observations for micro-units at specific points in space, i.e. spatial point patterns, and/or observations for aggregate spatial entities, i.e. area data.

Of note is that point data can be converted to area data, and area data can be represented by point reference. Areas may be regularly shaped such as pixels in remote sensing or irregularly shaped such as statistical reporting units. When divorced from their spatial context such data lose value and meaning. They may be viewed as single realisations of a spatial stochastic process, similar to the approach taken in the analysis of time series.

1.2 The Tyranny of Data

Analysing and modelling spatial data present a series of problems. Solutions to many of them are obvious, others require extraordinary effort for their solution. Data exercise a power that can lead to misinterpretation and meaningless results; therein lies the tyranny of data.

Quantitative analysis crucially depends on data quality. Good data are reliable, contain few or no mistakes, and can be used with confidence. Unfortunately, nearly all spatial data are flawed to some degree. Errors may arise in measuring both the location and attribute properties, but may also be associated with computerised processes responsible for storing, retrieving, and manipulating spatial data. The solution to the data quality problem is to take the necessary steps to avoid having faulty data determining research results.

The particular form [i.e. size, shape and configuration] of the spatial aggregates can affect the results of the analysis to a varying, usually unknown, degree as evidenced in various types of analysis (see, e.g., Openshaw and Taylor 1979, Baumann et al. 1983). This problem has become generally recognised as the *modifiable areal unit problem* (MAUP), the term stemming from the fact that areal units are not 'natural' but usually arbitrary constructs. For reasons of confidentiality, social science data (e.g., census data) are not often released for the primary units of observation (individuals), but only for a set of rather arbitrary areal aggregations (enumeration districts or census tracts). The problem arises whenever area data are analysed or modelled and involves two effects: one derives from selecting different areal boundaries while holding the overall size and the number of areal units constant (the zoning effect). The other derives from reducing the number but increasing the size of the areal units (the scale effect). There is no analytical solu-

tion to the MAUP, but questions of the following kind have to be considered in constructing an areal system for analysis: What are the basic spatial entities for defining areas? What theory guides the choice of the spatial scale? Should the definition process follow strictly statistical criteria and merge basic spatial entities to form larger areas using some regionalisation algorithms (see Wise et al. 1996)? These questions pose daunting challenges.

In addition, boundary and frame effects (i.e. the geometric structure of the study area) may affect spatial analysis and the interpretation of results. These problems are considerably more complex than in time series. Although several techniques, such as refined *K* function analysis, take the effect of boundaries into account, there is need to study boundary effects more systematically.

An issue that has been receiving increasing attention relates to the suitability of data. If the data, for example, are available only at the level of spatial aggregates, but the research question is at the individual respondent level, then the *ecological fallacy* (*ecological bias*) *problem* arises. Using area-based data to draw inferences about underlying individual–level processes and relationships poses considerable risks. This problem relates to the MAUP through the concept of spatial autocorrelation.

Spatial autocorrelation (also referred to as spatial dependence or spatial association) in the data can be a serious problem, rendering conventional statistical analysis unsafe and requiring specialised spatial analytical tools. This problem refers to situations where the observations are non-independent over space. That is, nearby spatial units are associated in some way. Sometimes, this association is due to a poor match between the spatial extent of the phenomenon of interest (e.g., a labour or housing market) and the administrative units for which data are available. Sometimes, it is due to a spatial spillover effect. The complications are similar to those found in time series analysis, but are exacerbated by the multidirectional, two-dimensional nature of dependence in space rather than the unidirectional nature in time. Avoiding the pitfalls arising from spatially correlated data is crucial to good spatial data analysis, whether exploratory or confirmatory. Several scholars even argue that the notion of spatial autocorrelation is at the core of spatial analysis (see, e.g., Tobler 1979). No doubt, much of current interest in spatial analysis is directly derived from the monograph of Cliff and Ord (1973) on spatial autocorrelation that opened the door to modern spatial analysis.

2 Pattern Detection and Exploratory Analysis

Exploratory data analysis is concerned with the search for data characteristics such as trends, patterns and outliers. This is especially important when the data are of poor quality or genuine *a priori* hypotheses are lacking. Many such techniques emphasise graphical views of the data that are designed to highlight particular features and allow the analyst to detect patterns, relationships, outliers etc. Exploratory spatial data analysis (ESDA), an extension of exploratory data analysis

(EDA) (Haining 1990, Cressie 1993), is especially geared to dealing with the spatial aspects of data.

2.1 Exploratory Techniques for Spatial Point Patterns

Point patterns arise when the important variable to be analysed is the location of events. At the most basic level, the data comprise only the spatial coordinates of events. They might represent a wide variety of spatial phenomena such as, cases of disease, crime incidents, pollution sources, or locations of stores. Research typically concentrates on whether the proximity of particular point events, their location in relation to each other, represents a significant (i.e., non-random) pattern. Exploratory spatial point pattern analysis is concerned with exploring the first and second order properties of spatial point pattern processes. First order effects relate to variation in the mean value of the process (a large scale trend), while second order effects result from the spatial correlation structure or the spatial dependence in the process.

Three types of methods are important: Quadrat methods, kernel estimation of the intensity of a point pattern, and distance methods. *Quadrat methods* involve collecting counts of the number of events in subsets of the study region. Traditionally, these subsets are rectangular (thus the name quadrat), although any shape is possible. The reduction of complex point patterns to counts of the number of events in quadrats and to one-dimensional indices is a considerable loss of information. There is no consideration of quadrat locations or of the relative positions of events within quadrats. Thus, most of the spatial information in the data is lost. Quadrat counts destroy spatial information, but they give a global idea of subregions with high or low numbers of events per area. For small quadrats more spatial information is retained, but the picture degenerates into a mosaic with many empty quadrats.

Estimating the intensity of a spatial point pattern is very like estimating a bivariate probability density, and bivariate *kernel estimation* can easily be adapted to give an estimate of intensity. Choice of the specific functional form of the kernel presents little practical difficulty. For most reasonable choices of possible probability distributions the kernel estimate will be very similar, for a given bandwidth. The bandwidth determines the amount of smoothing. There are techniques that attempt to optimise the bandwidth given the observed pattern of event location.

A risk underlying the use of quadrats is that any spatial pattern detected may be dependent upon the size of the quadrat. In contrast, *distance methods* make use of precise information on the locations of events and have the advantage of not depending on arbitrary choices of quadrat size or shape. *Nearest neighbour methods* reduce point patterns to one-dimensional nearest neighbour summary statistics (see Dacey 1960, Getis 1964). But only the smallest scales of patterns are considered. Information on larger scales of patterns is unavailable. These statistics indicate merely the direction of departure from Complete Spatial Randomness (CSR). The empirical K function, a reduced second-moment measure of the

observed process, provides a vast improvement over nearest neighbour indices (see Ripley 1977, Getis 1984). It uses the precise location of events and includes all event-event distances, not just nearest neighbour distances, in its estimation. Care must be taken to correct for edge effects. *K* function analysis can be used not only to explore spatial dependence, but also to suggest specific models to represent it and to estimate the parameters of such models. The concept of *K* functions can be extended to the multivariate case of a marked point process (i.e. locations of events and associated measurements or marks) and to the time-space case.

2.2 Exploratory Analysis of Area Data

Exploratory analysis of area data is concerned with identifying and describing different forms of spatial variation in the data. Special attention is given to measuring spatial association between observations for one or several variables. Spatial association can be identified in a number of ways, rigorously by using an appropriate spatial autocorrelation statistic (Cliff and Ord 1981), or more informally, for example by using a scatter-plot and plotting each value against the mean of neighbouring areas (Haining 1990).

In the rigorous approach to spatial autocorrelation the overall pattern of dependence in the data is summarised in a single indicator, such as Moran's *I* and Geary's *c*. While Moran's *I* is based on cross-products to measure value association, Geary's *c* employs squared differences. Both require the choice of a spatial weights or contiguity matrix that represents the topology or spatial arrangement of the data and represents our understanding of spatial association. Getis (1991) has shown that these indicators are special cases of a general formulation (called gamma) defined by a matrix representing possible spatial associations (the spatial weights matrix) among all areal units, multiplied by a matrix representing some specified non-spatial association among the areas. The non-spatial association may be a social, economic, or other relationship. When the elements of these matrices are similar, high positive autocorrelation arises. Spatial association specified in terms of covariances leads to Moran's *I*, specified in terms of differences, to Geary's *c*.

These global measures of spatial association can be used to assess spatial interaction in the data and can be easily visualised by means of a spatial variogram, a series of spatial autocorrelation measures for different orders of contiguity. A major drawback of global statistics of spatial autocorrelation is that they are based on the assumption of spatial stationarity, which implies *inter alia* a constant mean (no spatial drift) and constant variance (no outliers) across space. This was useful in the analysis of small data sets characteristic of pre-GIS times but is not very meaningful in the context of thousands or even millions of spatial units that characterise current, data-rich environments.

In view of increasingly data-rich environments a focus on local patterns of association (‘hot spots’) and an allowance for local instabilities in overall spatial association has recently been suggested as a more appropriate approach. Examples of techniques that reflect this perspective are the various geographical analysis

machines developed by Openshaw and associates (see, e.g., Openshaw et al. 1990), the Moran scatter plot (Anselin 1996), and the distance-based G_i and G_i^* statistics of Getis and Ord (1992). This last has gained wide acceptance. These G indicators can be calculated for each location i in the data set as the ratio of the sum of values in neighbouring locations [defined to be within a given distance or order of contiguity] to the sum over all the values. The two statistics differ with respect to the inclusion of the value observed at i in the calculation (included in G_i^*, not included in G_i). They can easily be mapped and used in an exploratory analysis to detect the existence of pockets of local non-stationarity, to identify distances beyond which no discernible association arises, and to find the appropriate spatial scale for further analysis.

No doubt, ESDA provides useful means to generate insights into global and local patterns and associations in spatial data sets. The use of ESDA techniques, however, is generally restricted to expert users interacting with the data displays and statistical diagnostics to explore spatial information, and to fairly simple low-dimensional data sets. In view of these limitations, there is a need for novel exploration tools sufficiently automated and powerful to cope with the data-richness-related complexity of exploratory analysis in spatial data environments (see, e.g., Openshaw and Fischer 1994).

3 Model Driven Spatial Data Analysis

ESDA is a preliminary step in spatial analysis to more formal modelling approaches. Model driven analysis of spatial data relies on testing hypotheses about patterns and relationships, utilising a range of techniques and methodologies for hypothesis testing, the determination of confidence intervals, estimation of spatial models, simulation, prediction, and assessment of model fit. Getis and Boots (1978), Cliff and Ord (1981), Upton and Fingleton (1985), Anselin (1988), Griffith (1988), Haining (1990), Cressie (1993), Bailey and Gatrell (1995) have helped to make model driven spatial data analysis accessible to a wide audience in the spatial sciences.

3.1 Modelling Spatial Point Patterns

Spatial point pattern analysis grew out of a hypothesis testing and not out of the pattern recognition tradition. The spatial pattern analyst tests hypotheses about the spatial characteristics of point patterns. Typically, Complete Spatial Random (CSR) represents the null hypothesis against which to assess whether observed point patterns are regular, clustered, or random. The standard model for CSR is that events follow a homogeneous Poisson process over the study region; that is, events are independently and uniformly distributed over space, equally likely to occur anywhere in the study region and not interacting with each other.

Various statistics for testing CSR are available. Nearest neighbour tests have their place in distinguishing CSR from spatially regular or clustered patterns. But little is known about their behaviour when CSR does not hold. The *K* function may suggest a way of fitting alternative models. Correcting for edge effects, however, might provide some difficulty. The distribution theory for complicated functions of the data can be intractible even under the null hypothesis of CSR. Monte Carlo tests is a way around this problem.

If the null hypothesis of CSR is rejected, the next obvious step in model driven spatial pattern analysis is to fit some alternative (parametric) model to the data. Departure from CSR is typically toward regularity or clustering of events. Clustering can be modelled through a heterogeneous Poisson process, a doubly stochastic point process, or a Poisson cluster process arising from the explicit incorporation of a spatial clustering mechanism. Simple inhibition processes can be utilised to model regular point patterns. Markov point processes can incorporate both elements through large-scale clustering and small-scale regularity. After a model has been fitted (usually via maximum likelihood or least squares using the *K* function), diagnostic tests have to be performed to assess its goodness-of-fit. Inference for the estimated parameters is often needed in response to a specific research question. The necessary distribution theory for the estimates can be difficult to obtain in which case approximations may be necessary. If, for example, clustering is found, one may be interested in the question whether particular spatial aggregations, or clusters, are associated with proximity to particular sources of some other factor. This leads to multivariate point pattern analysis, a special case of marked spatial point process analysis. For further details see Cressie (1993).

3.2 Modelling Area Data

Linear regression models constitute the leading modelling approach for analysing social and economic phenomena. But conventional regression analysis does not take into account problems associated with possible cross-sectional correlations among observational units caused by spatial dependence. Two forms of spatial dependence among observations may invalidate regression results: spatial error dependence and spatial lag dependence.

Spatial error dependence might follow from measurement errors such as a poor match between the spatial units of observation and the spatial scale of the phenomenon of interest. Presence of this form of spatial dependence does not cause ordinary least squares estimates to be biased, but it affects their efficiency. The variance estimator is downwards biased, thus inflating the R^2. It also affects the *t*- and *F*-statistics for tests of significance and a number of standard misspecification tests, such as tests for heteroskedasticity and structural stability (Anselin and Griffith 1988). To protect against such difficulties, one should use diagnostic statistics to test for spatial dependence among error terms and, if necessary, take action to properly specify the spatially autocorrelated residuals. Typically, dependence in the error term is specified as a spatial autoregressive or as a spatial moving avera-

ge process. Such regression models require nonlinear maximum likelihood estimation of the parameters (Cliff and Ord 1981, Anselin 1988).

In the second form, spatial lag dependence, spatial autocorrelation is attributable to spatial interactions in data. This form may be caused, for example, by significant spatial externalities of a socioeconomic process under study. Spatial lag dependence yields, biased and also inconsistent parameters. To specify a regression model involving spatial interaction, one must incorporate the spatial dependence into the covariance structure either explicitly or implicitly by means of an autoregressive and/or moving-average interaction structure. This constitutes the model identification problem that is usually carried out using the correlogram and partial correlogram. A number of spatial regression models, that is regression models with spatially lagged dependent variables, have been developed that include one or more spatial weight matrices which describe the many spatial associations in the data. The models incorporate either a simple general stochastic autocorrelation parameter or a series of autocorrelation parameters, one for each order contiguity (see Cliff and Ord 1981, Anselin 1988).

Maximum likelihood procedures are fundamental to spatial regression model estimation, but data screening and filtering can simplify estimation. Tests and estimators are clearly sensitive not only to the MAUP, but also to the specification of the spatial interaction structure represented by the spatial weights matrix. Recent advances in computation-intensive approaches to estimation and inference in econometrics and statistical modelling may yield new ways to tackle this specification issue. In practice, it is often difficult to choose between regression model specifications with spatially autocorrelated errors and regression models with spatially lagged dependent variables, though the 'common factor' approach (Bivand 1984) can be applied if the spatial lags are neatly nested.

Unlike linear regression, for which a large set of techniques for model specification and estimation now exist, the incorporation of spatial effects into nonlinear models in general – and into models with limited dependent variables or count data (such as log-linear, logit and tobit models) in particular – is still in its infancy. The hybrid log-linear models of Aufhauser and Fischer (1985) are among the few exceptions. Similarly, this is true for the design of models that combine cross-sectional and time series data for areal units. See Hordijk and Nijkamp (1977) for dynamic spatial diffusion models.

4 Toward Intelligent Spatial Analysis

Spatial analysis is currently entering a period of rapid change leading to what is termed intelligent spatial analysis [sometimes referred to as *GeoComputation*]. The driving forces are a combination of huge amounts of digital spatial data from the GIS data revolution (with 100,000 to millions of observations), the availability of attractive softcomputing tools, the rapid growth in computational power, and the new emphasis on exploratory data analysis and modelling.

Intelligent spatial analysis has the following properties. It exhibits computational adaptivity (i.e. an ability to adjust local parameters and/or global configurations to accommodate in response to changes in the environment); computational fault tolerance in dealing with incomplete, inaccurate, distorted, missing, noisy and confusing data, information rules and constraints; speed approaching human-like turnaround; and error rates that approximate human performance. The use of the term 'intelligent' is therefore closer to that in computational intelligence than in artificial intelligence. The distinction between artificial and computational intelligence is important because our semantic descriptions of models and techniques, their properties, and our expectations of their performance should be tempered by the kind of systems we want, and the ones we can build (Bezdek 1994).

Much of the recent interest in intelligent spatial analysis stems from the growing realisation of the limitations of conventional spatial analysis tools as vehicles for exploring patterns in data-rich GI (geographic information) and RS (remote sensing) environments and from the consequent hope that these limitations may be overcome by judicious use of computational intelligence technologies such as evolutionary computation (genetic algorithms, evolutionary programming, and evolutionary strategies) (see Openshaw 1994) and neural network modelling (see Fischer 1998). Neural network models may be viewed as nonlinear extensions of conventional statistical models that are applicable to two major domains: first, as universal approximators to areas such as spatial regression, spatial interaction, spatial choice and space-time series analysis (see, e.g., Fischer and Gopal 1994); and second, as pattern recognisers and classifiers to intelligently allow the user to sift through the data, reduce dimensionality, and find patterns of interest in data-rich environments (see, e.g. Fischer et al. 1997).

References

Anselin L. (1996): The Moran scatterplot as an ESDA tool to assess local instability in spatial association. In: Fischer M.M., Scholten H.J. and Unwin D. (eds.) *Spatial Analytical Perspectives on GIS*, Taylor & Francis, London, pp. 111-125

Anselin L. (1988): *Spatial Econometrics: Methods and Models*, Kluwer Academic Publishers, Dordrecht, Boston, London

Anselin L. and Florax R.J.G.M. (eds.) (1995): *New Directions in Spatial Econometrics*, Springer, Berlin, Heidelberg, New York

Anselin L. and Griffith D.A. (1988): Do spatial effects really matter in regression analysis? *Papers of the Regional Science Association* 65 (1), 11-34

Aufhauser E. and Fischer M.M. (1985): Log-linear modelling and spatial analysis, *Environment and Planning A* 17 (7), 931-951

Bailey T.C. and Gatrell A.C. (1995): *Interactive Spatial Data Analysis*, Longman, Essex

Baumann J.H., Fischer M.M. and Schubert U. (1983): A multiregional labour supply model for Austria: The effects of different regionalisations in multiregional labour market modeling, *Papers of the Regional Science Association* 52, 53-83

Bezdek J.C. (1994): What's computational intelligence. In: Zurada J.M., Marks II R.J. and Robinson C.J. (eds.) *Computational Intelligence: Imitating Life*, IEEE Press, Piscataway [NJ]

Bivand R. (1984): Regression modelling with spatial dependence: An application of some class selection and estimation methods, *Geographical Analysis* 16, 25-37

Cliff A.D. and Ord J.K. (1981): *Spatial Processes, Models & Applications*, Pion, London

Cliff A.D. and Ord J.K. (1973): *Spatial Autocorrelation*, Pion, London

Cressie N.A.C. (1993): *Statistics for Spatial Data*, John Wiley, Chichester [UK], New York

Dacey M.F. (1960): A note on the derivation of the nearest neighbour distances, *Journal of Regional Science* 2, 81-87

Fischer M.M. (1998): Computational neural networks – a new paradigm for spatial analysis, *Environment and Planning A* 30 (10), 1873-1891

Fischer M.M. and Getis A. (eds.) (1997): *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg, New York

Fischer M.M. and Gopal S. (1994): Artificial neural networks: A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34 (4), 503-527

Fischer M.M., Scholten H.J. and Unwin D. (eds.) (1996): *Spatial Analytical Perspectives on GIS*, Taylor & Francis, London

Fischer M.M., Gopal S., Staufer P. and Steinnocher K. (1997): Evaluation of neural pattern classifiers for a remote sensing application, *Geographical Systems* 4 (2), 195-223 and 233-234

Fotheringham S. and Rogerson P. (eds.) (1994): *Spatial Analysis and GIS*, Taylor & Francis, London

Getis A. (1991): Spatial interaction and spatial autocorrelation: A cross-product approach, *Papers of the Regional Science Association* 69, 69-81

Getis A. (1984): Interaction modelling using second-order analysis, *Environment and Planning A* 16 (2), 173-183

Getis A. (1964): Temporal land-use pattern analysis with the use of nearest neighbour and quadrat methods, *Annals of the Association of American Geographers* 54, 391-399

Getis A. and Boots B. (1978): *Models of Spatial Processes*, Cambridge University Press, Cambridge

Getis A. and Ord K.J. (1992): The analysis of spatial association by use of distance statistics, *Geographical Analysis* 24 (3), 189-206

Griffith D.A. (1988): *Advanced Spatial Statistics*: *Special Topics in the Exploration of Quantitative Spatial Data Series*, Kluwer Academic Publishers, Dordrecht, Boston, London

Haining R. (1990): *Spatial Data Analysis in the Social Sciences*, Cambridge University Press, Cambridge

Hordijk L. and Nijkamp P. (1977): Dynamic models of spatial autocorrelation, *Environment and Planning A* 9 (5), 505-519

Longley P.A. and Batty M. (eds.) (1996): *Spatial Analysis: Modelling in a GIS Environment*, GeoInformation International, Cambridge [UK]

Openshaw S. and Fischer M.M. (1994): A framework for research on spatial analysis relevant to geostatistical information systems in Europe, *Geographical Systems* 2 (4), 325-337

Openshaw S. and Taylor P. (1979): A million or so correlation coefficients: Three experiments on the modifiable areal unit problem. In: Bennett R.J., Thrift N.J. and Wrigley N. (eds.) *Statistical Applications in the Spatial Sciences*, Pion, London, pp. 127-144

Openshaw S., Cross A. and Charlton M. (1990): Building a prototype geographical correlates exploration machine, *International Journal of Geographical Information Systems* 4, 297-312

Ripley B.D. (1977): Modelling spatial patterns, *Journal of the Royal Statistical Society* 39, 172-212

Tobler W. (1979): Cellular geography. In: Gale S. and Olsson G. (eds.) *Philosophy in Geography*, Reidel, Dordrecht, pp. 379-386

Upton G. and Fingleton B. (1985): *Spatial Data Analysis by Example*, John Wiley, Chichester [UK], New York

Wise S., Haining R. and Ma J. (1996): Regionalisation tools for the exploratory spatial analysis of health data. In: Fischer M.M. and Getis A. (eds.) *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg, New York, pp. 83-100

3 Spatial Interaction Models and the Role of Geographic Information Systems

Many of the more sophisticated techniques and algorithms to process spatial data in spatial models are currently not or hardly available in GISystems. This raises the question of how spatial models should be integrated with GISystems. This chapter discusses possibilities and problems of interfacing spatial interaction models and GISystems from a conceptual rather than a technical point of view. The contribution illustrates that the integration between spatial analysis/modelling and GIS opens up tremendous opportunities for the development of new, highly visual, interactive and computational techniques for the analysis of spatial flow data. Using the Spatial Interaction Modelling [*SIM*] *software package as an example, the chapter suggests that in spatial interaction modelling GIS functionalities are especially useful in three steps of the modelling process: zone design, matrix building and visualisation.*

1 Introduction

The research traditions of spatial modelling and GISystems have generally developed quite independently of one another. The research tradition of spatial modelling lies in the heartland of quantitative geography and regional science. Since the 1950s, enormous strides have been made in developing models of spatial systems represented in diverse ways as points, areas and networks. A wide array of models now exist which vary greatly in their theoretical, methodological and technical sophistication and relevance. In the past two decades, many of these models have been adapted to policy contexts and have found some, albeit generally limited, use in decision making to solve spatial problems.

It would be impossible within the limited space available to do justice to the wide range of spatial model approaches and application domains in the social sciences. Thus we will be concentrating on one, but important category of generic spatial models, namely *spatial interaction models*. The description and prediction of spatial interaction patterns have been a major concern to geographers, planners, regional scientists and transportation scientists for many decades.

Spatial interaction can be broadly defined as the movement of people, commodities, capital and information over geographic space that result from a decision process (see Batten and Boyce 1986). The term thus encompasses such diverse behaviour as migration, travel-to-work, shopping, recreation, commodity flows, capital flows, communication flows (for example, telephone calls), airline pas-

senger traffic, the choice of health care services, and even the attendance at events such as conferences, cultural and sport events (Haynes and Fotheringham 1984). In each case, an individual trades off in some way the benefit of the interaction with the costs that are necessary in overcoming the spatial separation between the individual and his or her possible destination. It is the pervasiveness of this type of trade-off in spatial behaviour which has made spatial interaction modelling so important and the subject of intensive investigation in human geography and regional science (Fotheringham and O'Kelly 1989).

Mathematical models describing spatial interaction behaviour have an analytically rigorous history as tools to assist regional scientists, economic geographers, regional and transportation planners. The original foundations for modelling interaction over space were based on the analogous world of interacting particles and gravitational force, as well as potential effects and notions of market area for retail trade. Since that time, the *gravity model* has been extensively employed by city planners, transportation analysts, retail location firms, shopping centers, investors, land developers and so on, with important refinements relating to appropriate weights, functional forms, definitions of economic distance and transportation costs, and with disaggregations by route choice, trip type, trip destination conditions, trip origin conditions, transport mode, and so forth. The gravity model is one of the earliest spatial models and continues to be used and extended today. The reasons for these strong and continuing interests are easy to understand and stem from both theoretical and practical considerations.

Contemporary spatial theories have led to the emergence of two major schools of analytical thought: the *macroscopic* school based upon probability arguments and entropy maximising formulations (Wilson 1967) and the microscopic one corresponding to a behavioural or utility-theoretic approach (for an overview see Batten and Boyce 1986). The volume of research on spatial interaction analysis prior to the evolution and popularisation of GIS technology demonstrates clearly that spatial interaction modelling can be undertaken without the assistance of GIS technology. It is equally evident that GISystems have proliferated essentially as storage and display media for spatial data.

The aim of this chapter is to describe some features of the *spatial interaction modelling* (SIM) system which has been developed at the Department of Economic and Social Geography (see Fischer et al. 1996 for more details). The program is written in *C* and operates on SunSPARC stations. SIM is embracing the conventional types of (static) spatial interaction models including the unconstrained, attraction-constrained, production-constrained and doubly-constrained models with the power, exponential, Tanner or the generalised Tanner function. The estimation can be achieved by least squares or maximum likelihood. The system has a graphic user interface. The user has to specify the number of origins (up to 1,000), the number of destinations (up to 1,000), the model type, the separation function and the estimation procedure, and then to input distance and interaction data as well as data for the origin and destination factors. The data are entered on one logical record per origin-destination pair.

The software presently exists independently of any GISystem. We will discuss some possibilities and problems of interfacing SIM and GIS from a conceptual,

rather than a technical point of view. The integration between spatial analysis/ modelling and GIS opens up tremendous opportunities for the development of new, highly visual, interactive and computational techniques for the analysis of spatial flow data.

2 The Model Toolbox of the SIM System

The most general form of a spatial interaction model may be written (see, for example Wilson 1967, Alonso 1978, Sen and Sööt 1981) as

$$T_{ij} = V_i \, W_j \, F_{ij} \tag{1}$$

where V_i is called an origin factor (a measure of origin propulsiveness), W_j is called a destination factor (a measure of destination attractiveness), and F_{ij}, termed a separation factor, measures the separation between zones or basic spatial units i and j (i = 1, …, I; j = 1, …, J). T_{ij} is the expected or theoretical flow of people, goods, commodities etc. from i to j. Space is represented in a discrete rather than a continuous manner. Thus, the spatial dimension of Equation (1) is introduced implicitly by the separation matrix F_{ij} which may be square or rectangular.

2.1 Model Specification

The SIM toolbox encompasses the conventional types of spatial interaction models (the doubly constrained model, the attraction-constrained model, the production-constrained model and the unconstrained model) which can be derived from Equation (1).

The type of model to be used in any particular application context depends on the information available on the spatial interaction system. Suppose, for example, we are given the task of forecasting migration or traffic patterns and we know the outflow totals O_i, for each origin i and the inflow totals D_j, for each destination j. The appropriate spatial interaction model for this situation is the *production-attraction* (or doubly) *constrained* spatial interaction model which has the following form:

$$T_{ij} = A_i \, O_i \, B_j \, D_j \, F_{ij} \tag{2}$$

with

$$A_i = \frac{1}{\sum_j B_j \, D_j \, F_{ij}} \tag{3}$$

$$B_j = \frac{1}{\sum_i A_i \; O_i \; F_{ij}} \tag{4}$$

where A_i and B_j are origin-specific and destination-specific balancing factors which ensure that the model reproduces the volume of flow orginating at *i* and ending in *j*, respectively. This model type has been extensively used as a trip distribution model.

If only inflow totals, D_j, are known, then we need a spatial interaction model which is termed *attraction-constrained* and has the following form:

$$T_{ij} = V_i \; B_j \; D_j \; F_{ij} \tag{5}$$

with

$$B_j = \frac{1}{\sum_i V_i \; F_{ij}} \; . \tag{6}$$

This type of model can be used to forecast total outflows from origins. Such a situation might arise, for example, in forecasting the effects of a new industrial zone within a city or in forecasting university enrollment patterns.

The *production-constrained* spatial interaction model is useful in a situation where the outflow totals are known. The form of this model type is:

$$T_{ij} = A_i \; O_i \; W_j \; F_{ij} \tag{7}$$

$$A_i = \frac{1}{\sum_j W_j \; F_{ij}} \; . \tag{8}$$

This model type can, for example, be used to forecast the revenues generated by particular shopping locations. The models (5) and (7) are usually referred to as *location models*, since by summing the model equations over the constrained subscripts, the amount of activity located in different zones can be calculated.

Suppose that apart from an accurate estimate of the total number of interactions in a system we have no other information available to forecast the spatial interaction pattern in the system. Then the *unconstrained* spatial interaction model is the appropriate model type. It has the following form:

$$T_{ij} = K \; V_i^{\mu} \; W_j^{\nu} \; F_{ij} \tag{9}$$

where μ reflects the relationship between T_{ij} and V_i, ν reflects the relationship between T_{ij} and W_j, and *K* denotes a scale parameter.

The models presented in Equations (1)-(9) are in a generalised form and no mention has yet been made of the functional form of the separation factor F_{ij}. The rather general form as implemented in the SIM toolbox is based on a vector-valued separation measure $d_{ij} = ({}^1d_{ij}, ..., {}^Kd_{ij})$ and following Sen and Pruthi (1983) defined as

$$F_{ij} = \exp\left[\sum_k \Theta_k {}^k d_{ij}\right] \tag{10}$$

where the ${}^k d_{ij}$ are different measures of separation from i to j, for example, distance, travel time or costs, and are assumed to be known. $\Theta = \{\Theta_1, ..., \Theta_k\}$ is the (unknown) separation function parameter vector. Equation (10) is sufficiently general for most practical purposes. For $\Theta_1 = \alpha$ and ${}^1d_{ij} = \ln d_{ij}$ it subsumes the *power* function

$$F_{ij} = d_{ij}^{\alpha} \tag{11}$$

for $\Theta_1 = \beta$ and ${}^1d_{ij} = d_{ij}$ the *exponential* function

$$F_{ij} = \exp(\beta\, d_{ij}) \tag{12}$$

for $\Theta_1 = \alpha$, $\Theta_2 = \beta$, ${}^1d_{ij} = \ln d_{ij}$ and ${}^2d_{ij} = d_{ij}$ the *Tanner* (or *gamma*) function

$$F_{ij} = d_{ij}^{\alpha} \exp(\beta\, d_{ij}) \tag{13}$$

and for $\Theta_1 = \alpha$, $\Theta_2 = \beta$, ${}^1d_{ij} = \ln {}^1d_{ij}$ and ${}^2d_{ij} = {}^2d_{ij}$ the *generalised Tanner* function

$$F_{ij} = {}^1d_{ij}^{\alpha} \exp(\beta\, {}^2d_{ij}) \tag{14}$$

only in the case of ML estimation. The SIM toolbox combines these four separation functions with the four conventional types of spatial interaction model. Common to all these models is the need to obtain estimates of their parameters.

3 Calibrating Spatial Interaction Models in the SIM System

The process of estimating the parameters of a relevant model is called model calibration. The SIM system provides the choice of two principally different calibration methods using regression or maximum likelihood.

3.1 Regression Method: Ordinary and Weighted Least Squares

For the regression method the spatial interaction models have first to be linearised. Then the parameter values are computed to minimise the sum of squared deviations between the estimated and observed flows. The *unconstrained* model (9) can easily be linearised using direct logarithmic transformation, while the *constrained* models (2)-(8) are intrinsically nonlinear in their parameters. To linearise the constrained models we use the *odds ratio* technique described by Sen and Sööt (1981), but in contrast to Sen and Pruthi (1983) for the more general case of rectangular origin/destination matrices. This technique separates the estimation of the separation function parameters from the calculation of the balancing factors and involves taking ratios of interactions so that the $A_i O_i$ and/or the $B_i D_i$ terms in the models cancel out.

We will briefly illustrate the basics of this technique for the doubly-constrained model (8) with the general separation function (10). The procedure uses the odds ratio $(T_{ij} / T_{ii})\,(T_{ji} / T_{jj}) = (F_{ij} / F_{ii})\,(F_{ji} / F_{jj})$ to produce the following linear version of the attraction-production-constrained model:

$$t_{ij} - t_{i\bullet} - t_{\bullet j} + t_{\bullet\bullet} = \sum_k \Theta_k ({}^k d_{ij} + {}^k d_{i\bullet} - {}^k d_{\bullet j} - {}^k d_{\bullet\bullet}) \tag{15}$$

where t stands for the natural logarithm of T and the subscript dot indicates that a mean has been taken with respect to the subscript replaced by the dot.

The problem of estimating the parameter vector Θ is then a problem of minimising the following objective function (the sum of the squared deviations between observations and predictions):

$$\sum_i \sum_j \left[t_{ij} - t_{i\bullet} - t_{\bullet j} + t_{\bullet\bullet} - \sum_k \Theta_k \, ({}^k d_{ij} + {}^k d_{i\bullet} - {}^k d_{\bullet j} - {}^k d_{\bullet\bullet}) \right]^2 \tag{16}$$

with respect to $\Theta_k, k = 1, ..., K$. In order to find a set of K parameters which minimise (16), the corresponding linear set of K normal equations with K unknown parameters has to be solved:

$$\sum_i \sum_j {}^k X_{ij}\ {}^1 X_{ij}\ \Theta_1 + \sum_i \sum_j {}^k X_{ij}\ {}^2 X_{ij}\ \Theta_2 + \ldots$$

$$+ \sum_i \sum_j {}^k X_{ij}\ {}^k X_{ij}\ \Theta_K \ = \ \sum_i \sum_j y_{ij}\ {}^k X_{ij} \tag{17}$$

with $y_{ij} := t_{ij} - t_{i\bullet} - t_{\bullet j} + t_{\bullet\bullet}$ and ${}^k X_{ij} = {}^k d_{ij} - {}^k d_{i\bullet} - {}^k d_{\bullet j} + {}^k d_{\bullet\bullet}$ for $k = 1, \ldots, K$. This set of linear equations is solved in SIM by decomposing the coefficient matrix, breaking up the set into two successive sets and employing forward and backward substitution. In the univariate case $K = 1$, for example, we obtain the following parameter estimate:

$$\Theta_1 = \frac{\sum_i \sum_j y_{ij}\ {}^1 X_{ij}}{\sum_i \sum_j {}^1 X_{ij}\ {}^1 X_{ij}} \,. \tag{18}$$

Once the separation function parameters have been estimated, the balancing factors A_i and B_i can be obtained by iterating (3) and (4).

In addition to ordinary least squares estimation, the SIM package also provides the option of weighted least squares estimation. Weighted least squares with the weight being

$$(T_{ij}^{-1} + T_{ji}^{-1} + T_{ii}^{-1} + T_{jj}^{-1})^{-0.5} \tag{19}$$

may be preferable to ordinary least squares to counteract the heteroscedastic error terms caused by logarithmic transformation (see Sen and Sööt 1981). The weighted least squares procedure implemented takes the underestimation of the constant term of the unconstrained model into account (see Fotheringham and O'Kelly 1989).

3.2 Maximum Likelihood Estimation: Principle and Algorithm

Maximum likelihood (ML) methods have been used for some time as useful and statistically sound methods for calibrating spatial interaction models (see Batty and Mackie 1972). We developed a method of this kind based on the simulated annealing approach combined with a modification of the downhill simplex method.

The steps involved in ML estimation include identifying a theoretical distribution for the interactions, maximising the likelihood function of this distribution with respect to the parameters of the interaction model, and then deriving equations which ensure the maximisation of the likelihood function. For convenience, the logarithm of the likelihood function is used since it is at a maximum whenever the likelihood function is at a maximum. Parameter estimates that maximise the

likelihood function are termed maximum likelihood estimates. They have several desirable properties, i.e. they are consistent, asymptotically efficient and asymptotically normally distributed. The method of obtaining ML estimates will be described in terms of the unconstrained model only.

Let T_{ij} denote a flow of persons from i to j. T_{ij} might be considered to be the outcome of a Poisson process if it is assumed that there is a constant probability of any individual in i moving to j, that the population of i is large, and the number of individuals interacting is an independent process (Flowerdew and Aitkin 1982, Fotheringham and O'Kelly 1989). Then the probability that T_{ij} is the number of people recorded as moving from i to j is given by

$$P(T_{ij}) = \frac{\exp(-T_{ij})\, T_{ij}^{T'_{ij}}}{T'_{ij}!} \tag{20}$$

where T_{ij} is the expected outcome of the Poisson process and T'_{ij} the observed value which is subject to sampling and measuring errors and thus fluctuates around the expected value. T_{ij} has to be estimated from (9). Then the following log-likelihood equation can be built:

$$L = \sum_i \sum_j (-T_{ij} + T'_{ij} \ln T_{ij} - \ln T'_{ij}!) \,. \tag{21}$$

Since T'_{ij} is given, $\ln T'_{ij}!$ can be ignored in the maximisation, and L will be at a maximum when

$$Z = \sum_i \sum_i (T'_{ij} \ln T_{ij} - T_{ij}) \tag{22}$$

is at a maximum. In the SIM system the method of simulated annealing (originally developed by Kirkpatrick et al. 1983) and a variant of the Metropolis algorithm proposed by Metropolis et al. (1953) is used in combination with a modification of the downhill simplex method of Nelder and Mead (1965) to maximise Z or equivalently to minimise $U := -Z$, i.e. to find the ML estimates of the relevant model.

The method of simulated annealing has attracted significant attention as suitable for optimisation problems of large scale, especially those where a desired global extremum is hidden among many poorer local extrema. At the heart of the method is an analogy with thermodynamics, specifically with the way that liquids freeze and crystallise, or metals cool and anneal. The essence of the process is slow cooling, allowing ample time for redistribution of the atoms as they lose mobility. This is the technical definition of annealing and essential for guaranteeing that a low energy state will be achieved (see, e.g. Otten and van Ginneken 1989).

Simulated annealing is essentially a stochastic global minimisation approach or, in other words, a more sophisticated stochastic procedure for performing hill climbing necessary to avoid getting stuck in local minima. It starts from an initial point

in the parameter space, takes a step, and evaluates the function U once. When minimising U, it accepts every downhill movement and repeats the process from this new starting point. It may accept an uphill movement and in doing so escape from local minima. Uphill movement occurs in a controlled fashion so that there is no danger of jumping out of a local minimum and following into a worse one. As the minimisation process continues, the step length decreases, and the probability of accepting uphill movements decreases as well. The search converges to a local – and sometimes global – minimum. The main drawback of the calibration procedure is long processing time, since it is necessary to perform a large number of random searches at each temperature step to arrive near the equilibrium state. The downhill simplex method serves to improve the performance of the simulated annealing algorithm in situations where local downhill moves exist.

The method requires the specification of four elements: the choice of the initial value of a control parameter (analogous to temperature with annealing in thermodynamics), an annealing schedule by which the control parameter is gradually reduced, the choice of the maximum loop allowed at each temperature, and initial values of the $(n + 1)$ n simplex matrix where n denotes the number of parameters of the model. If the control parameter is reduced sufficiently slowly by the chosen annealing schedule, it becomes likely that the simplex will shrink into the region containing the lowest relative minimum encountered. If the annealing schedule is too slow, a satisfactory solution might never be reached; if it is too fast, a premature convergence to a local minimum might occur. Thus, the choice of the annealing schedule is a rather critical element in this calibration approach and a subject of further research (for more details see, for example, van Laarhoven and Aarts 1988).

4 Interfacing SIM and GIS: Possibilities and Problems

A GIS may be viewed as a tool for supporting a wide range of spatial analysis techniques including primitive geometric operations such as computing the centroids of polygons or building buffers around lines, and more complex operations such as determining the shortest-path in a network. The analytical functionality of leading GISystems continues to grow. But the comparative simplicity of the analytical functionality and the lack of advanced modelling functionalities remains to be a major concern and are a familiar complaint in the literature (see Goodchild 1987, 1992, Burrough 1990, Openshaw 1990, Anselin and Getis 1992, Fischer and Nijkamp 1992, Anselin et al. 1993, Densham 1994). Much of the discussion on this topic has been conceptual in nature, with remarkably little effect and progress in incorporating the range of known techniques of spatial analysis and models into current systems. There are several reasons for this. The most obvious is the strong emphasis in the GIS market on information management with its comparatively unsophisticated analytical needs rather than on spatial analysis in the wider sense.

The integration of modelling capabilities and GIS can be achieved in several ways. Two stand out on a continuum which spans the range from loosely coupled to strongly coupled systems (Batty 1992, 1993). Simply transferring data via files from a model to a GIS rather than using data structures in shared memory is the strategy of loose coupling (Densham 1994). This is the current practice of linking GIS and modelling capabilities. Some progress has been made where an independent spatial analysis module relies on a GIS for its input data, and for such functions as display but there is still a lack of effective forms of strong coupling where data could be passed between a GIS and a spatial model without loss of higher structures such as topology, object data, metadata or various kinds of relationships. This is to a large extent owing to a lack of standards for data models (Goodchild 1992).

Strategies for strong coupling might range from embedding spatial models in their entirety within GIS (a strategy followed by some GIS vendors in order to add model based functionalities to their systems) to embedding selected GIS functionalities within the model system (a strategy followed by Birkin et al. 1990 and Wegener and Spiekermann 1996). From a modeller's point of view there is a strong argument which suggests that embedding spatial models into a GIS is wasteful in terms of employing the many overheads of a GIS which are never used in the modelling process. In addition, proprietary GIS are generally so inflexible that their functions cannot be easily dissembled from the packages (Batty and Xie 1994a).

The SIM software currently exists independent of a GISystem. There are several good reasons to loosely couple SIM with a geographic information system or to enrich SIM with GIS functionalities. These centre around three steps in the modelling process: zone design, matrix building and visualisation.

4.1 Zone Design

The decision on the selection and design of an appropriate spatial framework is a crucial issue in the model building process which affects both the interpretation and acceptability of the models (Openshaw 1977a, Baumann et al. 1988). GIS offers spatial (dis)aggregation and interactive graphics display capabilities which, combined with basic spatial data analysis, provide considerable power to design zoning systems meeting specific requirements (see Batty and Xie 1994a) and to analyse ecological fallacies and the modifiable areal unit problem and the significance of zoning system (i.e. aggregation and scale) effects on the chosen spatial interaction model. At present, the principal difficulties in designing an appropriate spatial framework are the absence of zone-design tools in GIS and the need to have an explicit rationale for the zone design process considered to be best for the problem at hand. Openshaw and Schmidt (1996) suggest fast parallel algorithms for the zone design process, among them the AZP algorithm of Openshaw (1977b) and a simulated annealing version. They do not ‘solve’ the modifiable areal unit problem, but at least bring the problem under user control.

4.2 Matrix Building

Unless the spatial separation or distance matrix is output of a transport model, it is time-consuming to create and update the d_{ij}, for example in terms of road distances, manually for a larger system. This task can be accomplished within a GIS that has already stored the locational and topological relationships between polygons and points using a network algorithm such as NETWORK in ARC/INFO. Producing an interzonal distance matrix is then a matter of selecting for each zone a centroid node.

4.3 Visualisation

The real power of GIS resides in their display facilities. The basic mode of representation of model results and predictions centres around the maps, although other forms of graphics are useful to visualise the inputs, performance (for example via linked windows for residual analysis) and operation of the models (see Batty 1992). Graphical representation of spatial relationships is generally more easily interpreted than numerical output. One way of displaying spatial flows on a map is to draw line segments between each pair of centroids. To show the volumes of flows, the segments may be coloured, for example, or drawn with varying thicknesses. Since flows are directional, it is necessary to show them in both directions using arrows.

The problems lie not only in the absence of interactive functions of current GIS such as zooming and the way these are controlled, but also, more fundamentally, in the number of zones. The map representation fails if the number of zones is large because the display becomes too cluttered. For a spatial interaction system with 100 zones, for example, there are 9,900 directional node pairs. There are two possible solutions to resolve the map clutter problem: first, to apply interactive parameter manipulations such as thresholding and filtering to reduce the visual complexity, and second, to use visual matrix representation where the links are represented by squares tiled on the display, but the easy interpretation and context provided by the spatial map is lost.

5 Conclusions and Outlook

The SIM sytem briefly described in this chapter has the following features. Via a graphical user interface it allows specification of the model type (unconstrained, attraction-constrained, production-constrained, doubly-constrained), the spatial separation function (power, exponential, Tanner, generalised Tanner), the estimation procedure (ordinary/weighted least squares or maximum likelihood) to predict flows within a spatial interaction system with up to 1,000 origins and up to 1,000 destinations. To measure the performance of the chosen model, SIM provides the user with goodness-of-fit statistics including standardised root mean square, stan-

dardised R^2, information gain or likelihood ratio. The program was written in *C* by Jinfeng Wang and Adrian Trapletti using the Motif toolkit of the Open Software Foundation (OSF) for SunSPARC workstations.

GISystems are in essence a storage and display medium for spatial data, and if spatial interaction processes have to be modelled, such modelling must typically be achieved outside the GIS. This leads to the question of combining the strengths of the two systems. The simplest way of interfacing is loose coupling, i.e. to make use of the strength of a GIS for the purpose of spatial interaction modelling. The advantages to be gained from loose coupling include flexibility in the selection and design process of a spatial framework, saving time in creating and updating the spatial separation matrix and gaining greater display power for representing the model results and predictions in form of a map rather than a matrix. These benefits have to be weighed against the comparative slowness owing to the computational overhead of GIS. We share Densham's (1994) view that to develop effective embedded systems with reasonable expenditure of time and effort, GIS must provide better tools for integrating models, including direct user access to data structures.

This chapter has been concerned with static spatial interaction models. The extension to dynamic spatial interaction models would open up a new dimension of model use and application through spatial animation in presentation and communication. The next decade is likely to see new styles of spatial interaction modelling based on computational intelligence technologies such as evolutionary computation and neurocomputing.

***Acknowledgement*:** This chapter is based on research being carried out at the Vienna University of Economics and Business Administration supported by a grant from the Austrian Fonds zur Förderung der wissenschaftlichen Forschung (P 09972-TEC).

References

Alonso W. (1978): A theory of movement. In: Hansen N.M. (ed.) *Human Settlement Systems*, Ballinger, Cambridge, [MA], pp. 197-212

Anselin L. and Getis A. (1992): Spatial statistical analysis and geographic information systems, *The Annals of Regional Science* 26, 19-33

Anselin L., Dodson R.F. and Hudak S. (1993): Linking GIS and spatial data analysis in practice, *Geographical Systems* 1, 3-23

Baumann J.H., Fischer M.M. and Schubert U. (1988): A choice-theoretical labour-market model: Empirical tests at the mesolevel, *Environment and Planning A* 20 (8), 1085-1102

Batten D.F. and Boyce D.E. (1986): Spatial interaction, transportation, and interregional commodity flow models. In: Nijkamp P. (ed.) *Handbook of Regional and Urban Economics. Volume 1. Regional Economics*, North-Holland, Amsterdam, pp. 357-406

Batty M. (1993): Using geographic information systems in urban planning and policy-making. In: Fischer M.M. and Nijkamp P. (eds.) *Geographic Information Systems, Spa-*

tial Modelling, and Policy Evaluation, Springer, Berlin, Heidelberg, New York, pp. 51-69

Batty M. (1992): Urban models in computer-graphic and geographic information system environments. Paper presented at the Specialist Meeting, NCGIA, Initiative 14: GIS and Spatial Analysis, San Diego, 16-18 April 1992

Batty M. and Mackie S. (1972): The calibration of gravity, entropy and related models of spatial interaction, *Environment and Planning A* 4 (2), 205-233

Batty M. and Xie Y. (1994a): Modelling inside GIS: Part 1. Model structures, exploratory spatial data analysis and aggregation, *International Journal of Geographical Information Systems* 8 (3), 291-307

Batty M. and Xie Y. (1994b): Modelling inside GIS: Part 2. Selecting and calibrating urban models using ARC-INFO, *International Journal of Geographical Information Systems* 8 (5), 451-470

Birkin M., Clark G., Clarke M. and Wilson A.G. (1990): Elements of a model-based geographic information system for the evaluation of urban policy. In: Worrall L. (ed.) *Geographic Information Systems: Developments and Applications*, Belhaven Press, London, pp. 132-162

Burrough P. (1990): Methods of spatial analysis in GIS, *International Journal of Geographical Information Systems* 4, 221-223

Densham P.J. (1994): Integrating GIS and spatial modelling: Visual interactive modelling and location selection, *Geographical Systems* 1, 203-219

Fischer M.M. and Nijkamp P. (1992): Geographic information systems and spatial analysis, *The Annals of Regional Science* 26, 3-17

Fischer M.M., Trapletti A. and Wang J. (1996): SIM User's Manual. A flexible toolbox for spatial interaction modelling. WSG-RR-6, Department of Economic and Social Geography, Vienna University of Economics and Business Administration

Flowerdew R. and Aitkin M. (1982): A method of fitting the gravity model based on the Poisson distribution, *Journal of Regional Science* 22, 191-202

Fotheringham A.S. and O'Kelly M.E. (1989): *Spatial Interaction Models: Formulations and Applications,* Kluwer Academic Publishers, Dordrecht, Boston, London

Goodchild M. (1992): Geographical information science, *International Journal of Geographical Information Systems* 6, 31-45

Goodchild M. (1987): A spatial analytical perspective on geographical information systems, *International Journal of Geographical Information Systems* 1, 327-334

Haynes K.E. and Fotheringham S. (1994): *Gravity and Spatial Interaction Models*, Sage Publications, Beverly Hills, London and New Delhi, (= Scientific Geography Series, vol. 2)

Kirkpatrick S., Gelatt C.D. and Vecchi M.P. (1983): Optimization by simulated annealing, *Science* 220, 671-680

Laarhoven P.J.M. van and Aarts E.H.L. (1988): *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Dordrecht, Boston, London

Metropolis N., Rosenbluth A., Rosenbluth M., Teller A. and Teller E. (1953): Equations of state calculations by fast computing machines, *Journal of Chemical Physics* 21, 1087-1092

Nelder J.A. and Mead R. (1965): The downhill simplex method, *Computer Journal* 7, 308-313

Openshaw S. (1990): Spatial analysis and geographical information systems: A review of progress and possibilities. In: Scholten H.J. and Stillwell J.C.M. (eds.) *Geographical Information Systems for Urban and Regional Planning*, Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 153-163.

Openshaw S. (1977a): Optimal zoning systems for spatial interaction models, *Environment and Planning A* 9 (2), 169-184

Openshaw S. (1977b): A geographical solution to scale and aggregation problems in region-building, partitioning, and spatial modelling, *Transactions of the Institute of British Geographers*, *New Series* 2, 459-472

Openshaw S. and Schmidt J. (1996): Parallel simulated annealing and genetic algorithms for re-engineering zoning systems, *Geographical Systems* 3, 201-220

Otten R.H.J.M. and van Ginneken L.P.P.P. (1989): *The Annealing Algorithm,* Kluwer Academic Publishers, Dordrecht, Boston, London

Sen A. and Pruthi R.K. (1983): Least squares calibration of the gravity model when intrazonal flows are unknown, *Environment and Planning A* 15 (11), 1545-1550

Sen A. and Sööt S. (1981): Selected procedures for calibrating the generalised gravity model, *Papers of the Regional Science Association* 48, 165-176

Wegener M. and Spiekermann K. (1996): Efficient, equitable and ecological urban systems. In: Hensher D.A. and King J. (eds.) *World Transport Research. Proceedings of the Seventh World Conference on Transport Research*, Vol. 2, Pergamon, Oxford

Wilson A.G. (1967): A statistical theory of spatial trip distribution models, *Transportation Research* 1, 253-269

4 GIS and Network Analysis

This chapter discusses data model and design issues that are specifically oriented to GIS-T, the application of GISystems to research, planning and management in transportation, and identifies several improvements of the traditional network data model that are required to support advanced network analysis in a ground transportation context. These improvements include turn-tables, dynamic segmentation, linear referencing, traffic lines and non-planar networks. The chapter shifts attention also to network routing problems that have become prominent in GIS-T: the traveling-salesman problem, the vehicle-routing problem and the shortest-path problem with time windows, a problem that occurs as a subproblem in many time-constrained routing and scheduling issues of practical importance.

1 Introduction

Both geographic information systems (GIS) and network analysis are burgeoning fields, characterised by rapid methodological and scientific advances in recent years. A GIS is a digital computer application designed for the capture, storage, manipulation, analysis and display of geographic information. Geographic location is the element that distinguishes geographic information from all other types of information. Without location, data are termed to be non-spatial and would have little value within a GIS. Location is, thus, the basis for many benefits of GIS: the ability to map, the ability to measure distances and the ability to tie different kinds of information together because they refer to the same place (Longley et al. 2001).

GIS-T, the application of geographic information science and systems to transportation problems, represents one of the most important application areas of GIS technology today. While the strengths of standard GIS technology are in mapping display and geodata processing, GIS-T requires new data structures to represent the complexities of transportation networks and to perform different network algorithms in order to fulfil its potential in the field of logistics and distribution logistics.

This chapter addresses these issues as follows. The section that follows discusses data models and design issues which are specifically oriented to GIS-T, and identifies several improvements of the traditional network data model that are needed to support advanced network analysis in a ground transportation context. These improvements include turn-tables, dynamic segmentation, linear referencing, traffic lines and non-planar networks. Most commercial GIS software ven-

dors have extended their basic GIS data model during the past two decades to incorporate these innovations (Goodchild 1998).

The third section shifts attention to network routing problems that have become prominent in GIS-T: the traveling-salesman problem, the vehicle routing problem and the shortest-path problem with time windows, a problem that occurs as a subproblem in many time constrained routing and scheduling issues of practical importance. Such problems are conceptually simple, but mathematically complex and challenging. The focus is on theory and algorithms for solving these problems. The chapter concludes with some final remarks.

2 Network Representation and GIS-T Network Data Models

2.1 Terminology

A network is referred to as a pure network if only its topology and connectivity are considered. If a network is characterised by its topology and flow characteristics (such as capacity constraints, path choice and link cost functions) it is referred to as a flow network. A transportation network is a flow network representing the movement of people, vehicles or goods (Bell and Iida 1997).

The approach adopted almost universally is to represent a transportation network by a set of nodes and a set of links. The nodes represent points in space and possibly also in time, and the links tend to correspond to identifiable pieces of transport infrastructure (like a section of road or railway). Links may be either directed, in which case they specify the direction of movement, or undirected.

In graph theoretical terminology, a transportation network can be referred to as a valued graph, or alternatively a network. Directed links are referred to as arcs, while undirected links as edges. Other useful terms with some intuitive interpretations are a path, which is a sequence of distinct nodes connected in one direction by links; a cycle, which is a path connected to itself at the ends; and a tree, which is a network where every node is visited once and only once. The relationship between the nodes and the arcs, referred to as the network topology, can be specified by a node-arc incidence matrix: a table of binary or ternary variables stating the presence or absence of a relationship between network elements. The node-arc incidence matrix specifies the network topology and is useful for network processing.

2.2 The Network Data Model

The heart of any GIS is its data model. A data model is an abstract representation of some real world situation used to organise data in a database. Data models typically consist of three major components. The first is a set of data objects or entity types that form the basic building blocks for the database. The second component

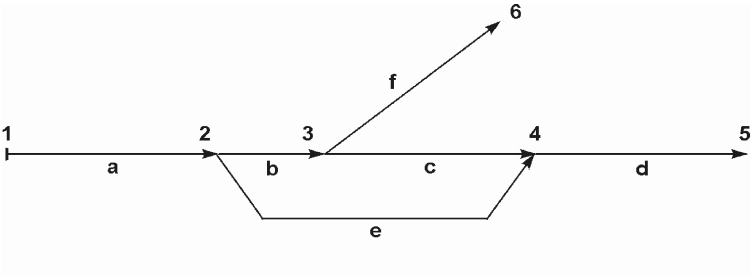
is a set of general integrity rules which constrain the occurrences of entities to those which can legally appear in the database. The final component includes operators that can be applied to entities in the database (Miller and Shaw 2001).

Data modelling involves three different levels of abstraction: conceptual, logical, and physical. Conceptual data models describe the organisation of data at a high level of abstraction, without taking implementation aspects into account. The entity-relationship and the extended entity-relationship models are the most widely used conceptual data models. They provide a series of concepts such as entities, relationships, and attributes, capable of describing the data requirements of an application in a manner that is easy to understand and independent of the criteria for managing and organising data on the system. A logical data model translates the conceptual model into a system-specific data scheme, while low-level physical data models provide the details of physical implementation (file organisation and indexes) on a given logical data model (Atzeni et al. 1999).

The network data model is the most popular conceptual model to represent a network within a GIS environment. The model – a special type of the node-arc-area data model that underlies many basic vector GIS databases – is built around two core entities: the node (a zero-dimensional entity) and the arc (a one-dimensional entity). Current GIS data models typically represent a network as a collection of arcs with nodes created at the arc intersections. The planar embedding of the node-arc data model guarantees topological consistency of the network.

The most widely used logical data model that supports the node-arc representation of networks is the georelational model. This model separates spatial and attribute data into different data models. A logical spatial data model (the vector data model) that encodes nodes and arcs maintains the geometry and associated topological information, while the associated attribute information is held in relational database management (RDBMS) tables. Unique identifiers associated with each spatial entity (node or arc) provide links to records in the relational model and its data on the attributes of the entity. This hybrid data management strategy was developed to take advantage of an RDBMS to store and manipulate attribute information (Longley et al. 2001). But this solution does not allow the relationships between a spatial object and its attributes to have their own attributes (Goodchild 1998). Though the solution is neither elegant nor robust, it is effective, and, the georelational model is widely present in GIS software (Miller and Shaw 2001).

The relational structure to support the planar network model typically consists of an arc relation and a node relation. The structure may be illustrated as a representation of the simple network shown graphically in Figure 1(a). The model implemented in GIS represents each arc of the network as a polyline entity. Associated with each entity will be a set of attributes, conceived as the entries in one row of a rectangular table (see Figure 1(b)). Properties may include information about the transverse structure such as the number of lanes or information on address locations within the network. Commonly included attributes are arc length, free flow travel time, base flow, and estimated flow. The base and estimated flows usually refer to the observed flow and the flow estimated from some modelling exercise (Miller and Shaw 2001).



(a) Example network for the relational model example

Arc ID	*Street name*	*Lanes*	*Other attributes*
a	High Street	2	
b	High Street	4	
c	High Street	4	
d	High Street	2	
e	River Way	2	
f	Hill Street	2	

(b) A simple arc table

Node ID	*Stop light*	*Other attributes*
1	n	
2	y	
3	n	
4	y	
5	n	
6	n	

(c) A simple node table

Arc ID	*Street name*	*Lanes*	*From node*	*To node*
a	High Street	2	1	2
b	High Street	4	2	3
c	High Street	4	3	4
d	High Street	2	4	5
e	River Way	2	2	4
f	Hill Street	2	3	6

Node ID	*Stop light*	*Arc links*
1	n	a
2	y	a, b, e
3	n	b, c, f
4	y	c, d, e
5	n	d
6	n	f

(d) Pointers added to the arc and node tables to represent connectivity

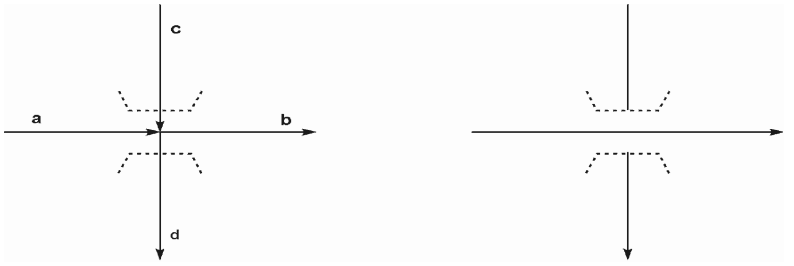
Figure 1 Relational data model representations of the arcs and nodes of a network: (a) example network for the relational model example, (b) a simple arc table, (c) a simple node table, (d) pointers added to the arc and node tables to represent connectivity (adapted from Goodchild 1998)

The node relation typically contains a node identification field and relevant attributes of the node, such as the presence of a traffic light. Figure 1(d) shows a scheme that includes storage of pointers from arcs to nodes, and from nodes to arcs, to store the topology of the network (connectivity). Each arc has an inherent direction defined by the order of points in its polyline representation, as illustrated by arrows in Figure 1(a) (Goodchild 1998). It is noteworthy that the node-arc representation disaggregates a transportation system into separate subnetworks for

each mode within a single base network. Transfer arcs link the subnetworks. The pseudo-arcs represent modal transfers.

2.3 Non-Planar Networks and the Turn-Table

The planar network data model has received widespread acceptance and use. Despite its popularity, the model has limitations for some areas of transportation analysis, especially where complex network structures are involved. One major problem is caused by the planar embedding requirement. This requirement forces nodes to exist at all arc intersections and, thus, ensures topological consistency of the model. But intersections at grade cannot be distinguished from intersections with an overpass or underpass that do not cross at grade. This difficulty in representing underpasses or overpasses may lead to problems when running routing algorithms (Kwan et al. 1996). The drawbacks in planar topology for network representations have motivated interest in non-planar network models. Planar models force nodes at all intersections (Figure 2(a)), while non-planar network models (Figure 2(b)) do not. Non-planar networks are broadly defined as those networks which permit arcs of the network to cross without a network node being located at the intersection. There is no implicit or explicit contact between the line segments at the point of intersections (Fohl et al. 1996).



(a) Planar network approach (b) Non-planar network approach

Figure 2 Illustration of (a) planar and (b) non-planar network approaches to represent an overpass

But non-planar data models provide only a partial solution to the problem of connectivity. In transportation network analysis it may be necessary to include extensive information on the ability to connect from one arc to another. Drivers, for example, may force turn restrictions or trucks may be limited by turning radius. Such situations require more than the simple ability to represent the existence of a crossing at grade or an underpass (Goodchild 1998).

To resolve this problem, the standard fully intersected planar network data model has been extended by adding a new structure, called the turn-table. Table 1 shows a turn-table for the layout used in Figure 2(a). For each ordered pair of arcs

incident at a node, a row of attributes in the table gives appropriate characteristics of the turn (yes/no), together with links to the tables that contain the attributes of the arcs. In this way, a data model with a planar embedding requirement can represent overpasses and underpasses by preventing turns (Goodchild 1998).

Table 1 Layout of a turn-table for the layout used in Figure 2(a)

From arc	*To arc*	*Turn ?*
a	c	n
a	b	y
a	d	n
b	a	y
b	c	n
b	d	n
c	a	n
c	b	n
c	d	y
d	a	n
d	b	n
d	c	y

2.4 Linear Referencing Systems and Dynamic Segmentation

While geographic features are typically located using planar referencing systems, many characteristics associated with a transportation network are located by means of a linear rather than coordinate-based system. These characteristics include data on transportation-related events and facilities (often termed feature data). In order to use linear-referenced attributes in conjunction with a spatially referenced transportation network, there must be some means of linking the two referencing systems together (Spear and Lakshmanan 1998).

Linear referencing systems typically consist of three components (Vanderohe and Hepworth 1996, Sutton 1997): a transportation network, a linear referencing method and a datum. The transportation network is represented by the conventional node-arc network. The linear referencing method determines an unknown location within the network using a defined path and an offset distance along that path from some known location (Miller and Shaw 2001). The datum is the set of objects (so-called reference or anchor points) with known georeferenced locations that can be used to anchor the distance calculations for the linear referenced objects.

There are different linear referencing methods. Nyerges (1990) identifies three major strategies, namely, road name and kilometerpoint (milepoint) referencing, control section locational referencing, and link and node locational referencing. Road name and kilometerpoint is a system familiar to anyone who has driven on highways in Europe or the USA. This system consists of a road-naming convention (i. e., a standard procedure for assigning names to highways and streets) and a series of kilometerpoint references (i. e., distance calculations along

the network, typically measured in fractions of a kilometer or mile). Kilometer-point referencing requires a designated point of reference (e. g., a kilometer 0) as a datum (see Figure 3). This is often an end point of the route or where the route crosses a provincial or a national boundary (Miller and Shaw 2001).

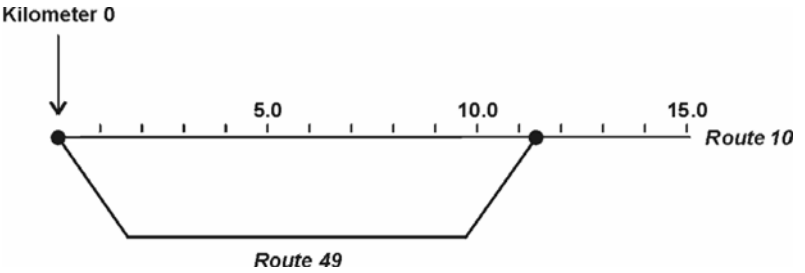
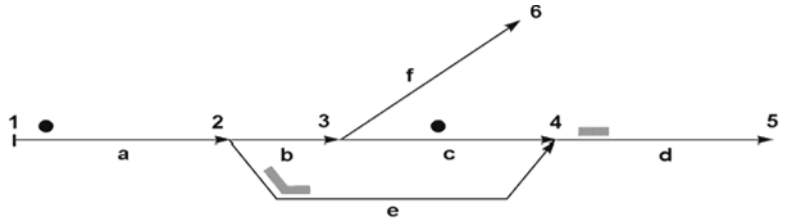


Figure 3 Kilometerpoint referencing

Owing to road modifications and other changes in road geometry, kilometerpoint referencing can become increasingly inaccurate over time. In other words, the reference kilometerpoint may not reflect the actual distance from the point of origin. This may cause problems when maintaining historical records of transportation events, and requires some type of translation factor to adjust distances (Nyerges 1990; Miller and Shaw 2001).



Arc	*Distance from arc start*	*Feature*
a	0.7	Bus depot
c	2.2	Bus depot

(a) Network points

Arc	*Distance from arc start to start of feature*	*Distance from arc start to end of feature*	*Level of congestion*
d	0.5	1.2	High
e	1.0	2.5	High

(b) Network segments

Figure 4 The concept of dynamic segmentation: A simple example for (a) network points and (b) network segments (adapted from Goodchild 1998)

The key to tie (zero-dimensional and one-dimensional) objects located at arbitrary locations on the network to the node-arc structure of the network data model is dynamic segmentation. The term derives from the fact that feature data values are held separately from the actual network route in database tables and then dynamically added to segments of the route each time the user queries the database (Longley et al. 2001). Several commercial GIS software packages provide dynamic segmentation capabilities, typically maintained at the logical level using the relational data model (Miller and Shaw 2001). Figure 4 provides a simple illustration of the concept, for two types of objects located at arbitrary locations on the network. These entities – termed network points (point events) and network segments (line events) – are given their own attribute tables. Dynamic segmentation reduces the number of transportation features or network links that have to be maintained to represent the system, and is particularly useful in situations in which the event data change frequently and need to be stored in a database for access by other applications (Longley et al. 2001).

2.5 Lanes and Navigable Data Models

A straightforward way to enhance the basic node-arc model for intelligent transportation systems (ITS) applications is to add information on the transverse structure of the network (Miller and Shaw 2001). Even though certain information about the transverse structure (such as the existence of a median or the number of lanes) might be stored as attributes of arcs or network lines, it is not possible to store detailed information about individual lanes or connectivity at the lane level. There is, for example, no way to disaggregate a turn-table to store turn restrictions which are specific to lanes (Goodchild 1998).

ITS database requirements go well beyond the traditional requirements of maintaining arc-node topology, two-dimensional georeferencing, and linear referencing of events within transportation networks. A fully fletched ITS requires a high-integrity, real time information system which will receive inputs from sensors embedded within transportation facilities and from vehicles equipped with Global Positioning System (GPS) devices and navigable data models. Navigable data models are digital geographic databases of a transportation system that can support vehicle guidance operations of different kinds. For intelligent transportation systems, this includes four functions (Dane and Rizos 1998; Miller and Shaw 2001):

- The data model has to unambiguously translate coordinate-based locations into street addresses, and vice versa. Travelers utilise address systems for location referencing while an ITS tracks a vehicle utilising a GPS receiver that can provide locations at accuracies of 5-10 m.

- The data model has to support map matching. This refers to the ability to snap the position of vehicle to the nearest location on a network segment when its estimated or measured location is outside the network. This may occur due to

differences in accuracy between the digital network database and the GPS system.

- The data model has to have the capability to represent the transportation network in detail sufficient to perform different network algorithms, modelling and simulations. In the real world, a transportation network has different types of intersections that are of interest to ITS builders. For some applications, information on intersections, lanes and lane changes, highway entrances and exits, etc., is important. Other applications may require geometric representation of road curvature and incline.

- The data model must not only assist the traveler in selecting an optimal route based on stated criteria such as travel time, cost and navigational simplicity but also support route guidance. This refers to navigational instructions and is a challenging task in real time.

Although dynamic segmentation can be used to enhance the traditional node-arc structure for ITS applications much of the high-resolution positional information provided by in-vehicle GPS receivers is lost when referenced within the traditional network structure. While 50 m accuracy may be sufficient to locate a vehicle on a road, better than 5 m will be required to locate to the lane level. Such accuracies are well beyond the capability of many of the currently available network databases. Achievement of better than 5 m accuracy using GPS requires the use of differential techniques and a high quality of geodetic control (Goodchild 1998).

Fohl et al. (1996) describe a prototype lane-based navigable data model where each lane is represented as a distinct entity, with its own connectivity with other lanes, but its geometry is obtained from the standard linear geometry of the road. No attempt is made to store the relative positions of lanes, but the structure does identify such topological properties as adjacency, and the order of lanes across the road (Goodchild 1998). A more radical approach to navigable data models for ITS is to abandon the node-arc model entirely. Bespalko et al. (1998) suggest a 3-dimensional object-oriented GIS-T data model that can distinguish between overpasses, underpasses and intersections, thereby providing guidance through complex intersections.

3 Vehicle Routing within a Network: Problems and Algorithms

At the core of many procedures in GIS-T software are algorithms for solving network-routing problems. The problems are conceptually simple but mathematically complex and challenging. How can we best route vehicles such as trucks, school buses, and general passenger buses from one location to another? The problems encountered in answering such questions have an underlying combinatorial

structure. For example, either we dispatch a vehicle or we do not, or we use one particular route or another.

This section deals with node-routing problems. Node-routing – in contrast to arc routing – refers to routing problems where the key service activity occurs at the nodes (customers), and arcs are of interest only as elements of paths that connect the nodes (Assad and Golden 1995). We discuss two specific problems that have become prominent in network analysis: the traveling-salesman problem and the vehicle-routing problem. The survey of basic network algorithms is completed by discussing a dynamic programming approach for solving the shortest-path problem with time windows, a problem that occurs as a subproblem in many time-constrained routing and scheduling issues.

At the outset of this section we should note that vehicle-routing algorithms can be applied in one of two modes: first, variable routing and second, fixed routing. In a variable-routing context, an algorithm is utilised with actual customer delivery requirements to develop routes for the next planning horizon. Fixed routing is applied when customer demands are sufficiently stable to allow use of the same routes repeatedly (Fisher 1995).

3.1 The Traveling-Salesman Problem

The simplest node-routing problem is the traveling-salesman problem. The traveling-salesman problem is a classical combinatorial optimisation problem that is simple to state but very difficult to solve. The problem is to find the least-cost tour through a set of nodes so that each node is visited exactly once. The tour starts and ends from a specific location, called depot. The problem is in a mathematical sense difficult, namely NP-complete (non-deterministic polynomial time-complete), and cannot be solved exactly in polynomial time. Although there are many ways to formally state the traveling-salesman problem, a convenient way in doing so is an integer linear programming formulation. Assume a directly connected network $G = (\boldsymbol{N}, \boldsymbol{A})$ where $\boldsymbol{N}$ is the node set with $| \boldsymbol{N} | = \boldsymbol{N}$ and $\boldsymbol{A}$ is the arc set defined as the Cartesian product of $\boldsymbol{N}$ with itself (that is, $\boldsymbol{A} = \boldsymbol{N} \times \boldsymbol{N}$), then the traveling-salesman problem can be formulated as

$$\min_{\{x_{ij}\}} \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij}\, x_{ij} \,, \tag{1}$$

subject to

$$\sum_{j=1}^{N} x_{ij} = 1 \qquad \text{for } i = 1, ..., N, \tag{2}$$

$$\sum_{i=1}^{N} x_{ij} = 1 \qquad \text{for } j = 1, ..., N, \tag{3}$$

$$(x_{ij}) \in \boldsymbol{X}, \tag{4}$$

$$x_{ij} \in \{0,1\} \qquad \text{for } i,\ j = 1, ..., N, \tag{5}$$

where c_{ij} is the length of the arc from node *i* to node *j*, and the x_{ij} are the decision variables: x_{ij} is set to 1 when arc (*i, j*) is included in the tour, and 0 otherwise. $(x_{ij}) \in \boldsymbol{X}$ denotes the set of subtour-breaking constraints that restrict the feasible solutions to those consisting of a single tour. The subtour-breaking constraints can be formulated in different ways. But one very intuitive formulation is

$$\sum_{i,\, j \in \boldsymbol{S_A}} x_{ijk} \leq |\boldsymbol{S_A}| - 1, \quad \boldsymbol{S_A} \subseteq \boldsymbol{A}, \quad 2 \leq |\boldsymbol{S_A}| \leq N - 2, \tag{6}$$

where $\boldsymbol{S_A}$ is some subset of $\boldsymbol{A}$ and $|\boldsymbol{S_A}|$ the cardinality of $\boldsymbol{S_A}$. These constraints prohibit subtours, i.e. tours on subsets with less than *N* nodes. If there were such a subtour on some subset of $\boldsymbol{S_A}$ of nodes, this subtour would contain $|\boldsymbol{S_A}|$ arcs. Consequently, the left-hand side of the inequality would be equal to $|\boldsymbol{S_A}|$, which is greater than $|\boldsymbol{S_A}| - 1$, and the constraint would be violated for this particular subset. Without expression (6), the traveling-salesman problem reduces to an assignment problem (Potvin 1993).

The traveling-salesman problem is a classically hard problem to solve optimally. Enumerating the possibilities works well for small *N*. But there are $(N - 1)!$ candidate itineries from which the single optimal one must be found. If $N = 100$, then the number of possible tours is 10^{200}. Many heuristic algorithms have been devised to solve the problem (for an overview see Lawler et al. 1985; Laporte 1992). These heuristics were designed to work quickly and to come close to the optimal solution. But they do not guarantee that the optimum will be found. Two broad classes of traveling-salesman heuristics can be distinguished: classical heuristic algorithms and optimisation-based algorithms.

Classical traveling-salesman heuristics include tour construction procedures, tour improvement procedures, and composite procedures that are based on both types of techniques. The best known tour construction heuristics gradually build up a tour by selecting each node in turn and inserting them one by one into the current tour. Various metrics may be utilised for the choice of the next node, such as proximity to the current tour. Among the tour improvement procedures, the *r*-opt exchange heuristics are the most widely used especially the 2-opt, 3-opt (see Lin 1965), and the interchange heuristic of Lin and Kernighan (1973). These traveling-salesman heuristics locally modify the current solution by replacing *r* arcs in the tour by *r* new arcs so as to generate a new improved tour. Characteristically, the exchange heuristics are applied iteratively until a local optimum is found, i.e. a tour that cannot be improved further via the exchange heuristic under consideration. To overcome the limitations associated with local optimality, new heuristics such as tabu search, simulated annealing and computational intelligence-based techniques may be utilised to escape from local minima (Kirkpatrick et al. 1983,

Glover 1989, 1990, Potvin 1993). Composite procedures make use of both tour construction and improvement techniques. They belong to the most powerful heuristics used to solve the traveling-salesman problem. The iterated Lin-Kernighan heuristic, for example, can routinely find solutions within 1 % of the optimum for traveling-salesman problems with up to 10,000 nodes (Johnson 1990).

Optimisation-based heuristics are very different in character from the classical traveling-salesman heuristics. They apply some optimisation algorithm and simply terminate prior to optimality. The most popular technique is branch-and-bound, originally applied to the traveling-salesman problem by Dantzig et al. (1954) and continually refined over the years. Branch-and-bound is a directed enumeration procedure that partitions the solution space into increasingly smaller subsets in an attempt to identify the subset that contains a near-optimal solution. For each subset a bound is calculated that estimates the best possible solution in the subset. The assignment problem relaxation, for example, may be used to generate lower bounds on the optimum. If the bound for a subset indicates that it cannot contain a near-optimal solution, the partitioning process is continued with another subset. The algorithm terminates when there are no subsets remaining. It is worth noting that traveling-salesman problems with a few hundred nodes can be routinely solved to optimality.

3.2 The Vehicle-Routing Problem

There are many ways of generalising the traveling-salesman problem to match real world situations. Often there is more than one vehicle, and in these situations the division of stops between variables is an important decision variable (Longley et al. 2001). In this section we consider the vehicle-routing problem. The problem is to route a fixed number of vehicles through a number of demand locations such that the total cost of travel is minimised and vehicle capacity constraints are not violated. Characteristically, there is a designated location known as the depot where all vehicles have to start and end their tours. There are numerous variations of this basic vehicle-routing problem, including time windows for delivery, stochastic demand, multiple depots with each vehicle in the fleet assigned to a particular depot etc. The problem and its extensions are of substantial practical importance and have resulted in the development of a great many heuristic algorithms including computational intelligence procedures over the past 35 years (for reviews see Bodin et al. 1983; Golden and Assad 1986; and Fisher 1995).

We view the vehicle-routing problem as consisting of two interlinked problems: first, finding an optimal assignment of customer orders to vehicles, and, second, ascertaining which route each vehicle will follow in servicing its assigned demand in order to minimise total delivery cost. To provide a precise statement of the problem we introduce notation first and then draw on Fisher and Jaikumar (1981) to specify the vehicle-routing problem as a nonlinear generalised assignment problem.

Let $\boldsymbol{N} = \{1, \ldots, N\}$ be the set of demand locations (customers) and $\boldsymbol{K} = \{1, \ldots, K\}$ be the set of available vehicles to be routed and scheduled. Consider

the network $G = (\boldsymbol{V}, \boldsymbol{A})$ where $\boldsymbol{V} = \boldsymbol{N} \cup \{0\}$ is the set of nodes with 0 representing the depot of the vehicles, and $\boldsymbol{A} = \boldsymbol{V} \times \boldsymbol{V}$ is the arc set that contains all arcs (i, j) with $i, j \in \boldsymbol{V}$. c_{ij} denotes the cost of direct travel from point i to point j, b_k the capacity (e. g., weight or volume) of vehicle k, and a_i the size of the order of customer $i \in \boldsymbol{N}$, measured in the same units as the vehicle capacity. Define the flow variable y_{ik} as $0 - 1$ variable equal to 1 if the order from customer i is delivered by vehicle k, and 0 otherwise, and $y_k := (y_{0k}, ..., y_{nk})$. Then the vehicle-routing problem can be formulated as the following nonlinear generalised assignment problem (Fisher 1995):

$$\min_{\{y_k\}} \sum_{k=1}^{K} f(y_k), \tag{7}$$

subject to

$$\sum_{i=1}^{N} a_i \, y_{ik} \le b_k \qquad \text{for } k = 1, ..., K, \tag{8}$$

$$\sum_{k=1}^{K} y_{ik} = \begin{cases} K & i = 0, \\ 1 & i = 1, ..., N, \end{cases} \tag{9}$$

$$y_{ik} \in \{0,1\} \qquad \text{for } i = 0, 1, ..., N; k = 1, ..., K. \tag{10}$$

Equations (8)-(10) are the constraints of the assignment problem and guarantee that each route begins and ends at the depot ($i = 0$), that each customer ($i = 1, \ldots, N$) is serviced by some vehicle, and that the load assigned to the vehicle is within its capacity. $f(y_k)$ represents the cost of an optimal traveling-salesman problem tour of all the points in $\boldsymbol{V}(y_k) = \{i \mid y_{ik} = 1\}$ that must be visited by each vehicle k to service its assigned customers. Defining $x_{ijk} = 1$ if vehicle k travels directly from i to j, and $x_{ijk} = 0$ otherwise, the function $f(y_k)$ can be defined mathematically as

$$f(y_k) = \min \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij} \, x_{ijk}, \tag{11}$$

such that

$$\sum_{i=1}^{N} x_{ijk} = y_{jk} \qquad \text{for } j = 0, ..., K, \tag{12}$$

$$\sum_{j=1}^{N} x_{ijk} = y_{ik} \qquad \text{for } i = 0, ..., K, \tag{13}$$

$$\sum_{(i,j)\in\{0,1\}} x_{ijk} \leq |\boldsymbol{S}| - 1, \quad \boldsymbol{S} \subseteq \boldsymbol{V}(y_k), \quad 2 \leq |\boldsymbol{S}| - \boldsymbol{N}. \tag{14}$$

$$x_{ijk} \in \{0,1\} \qquad \text{for } i = 0, ..., N;\ j = 1, ..., N. \tag{15}$$

The Fisher and Jaikumar (1981) method is the best-known heuristic to solve some mathematical programming approximation of the vehicle-routing problem to optimality. The heuristic replaces $f\ (y_k)$ with a linear approximation $\Sigma_i\, d_{ik}\, y_{ik}$ and solves the resulting linear generalised assignment problem to get an assignment of customers to vehicles (Fisher 1995). Once this assignment has been made, a complete solution is obtained by applying any traveling-salesman problem heuristic to get the delivery sequence for the customers assigned to each vehicle.

To obtain the linear approximation, Fisher and Jaikumar (1981) first specify K 'seed' customers $i_1, \ldots, i_K$ that are assigned one to each vehicle. Without loss of generality, customers i_k can be assigned to vehicle k for k = 1, …, K. Then the coefficient d_{ik} is set to the cost of inserting customer i into the route on which vehicle k travels from the depot directly to customer i_k and back. Specifically, $d_{ik} = c_{0i} + c_{iik} - c_{0ik}$. Clearly, the seed customers define the direction in which each vehicle will travel, and the assignment problem completes the assignment of customers to routes given this general framework (Fisher 1995). Seeds are generally chosen with the following rule. Choose the first seed s_1 to be a customer farthest away from the depot. If k seeds have been chosen, choose s_{k+1} to solve

$$\max_i \ \min \left\{ c_{i0}, \min_{j=1,\ldots,k} c_{is_j} \right\}. \tag{16}$$

The algorithm can be extended to accommodate a number of variations and generalisations of the above vehicle-routing problem such as the vehicle-routing problem with time windows. This problem consists of designing a set of minimum cost-routes, starting at and returning to a central depot for a fleet of vehicles that services a set of customers with known demands. The service at a customer has to begin within the time window defined by the earliest time and the latest time when the customer allows the start of the service. Time windows can be hard or soft. In the soft time windows case the time window case can be violated at a cost. In contrast, in the hard time window case a vehicle is not permitted to arrive at a node after the latest time to begin service. But, if a vehicle arrives too early at a node, it is allowed to wait until the node is ready for service to begin. The costs involved in time-constrained routing and scheduling consist of fixed-vehicle utilisation costs and variable routing and scheduling costs. The latter include distance

and travel time costs, waiting time costs, and loading/unloading time costs (Desrosiers et al. 1995).

3.3 Constrained Shortest-Path Problems

The section will be concluded by considering the shortest-path (or least-cost) problem with time windows. This problem appears as a subproblem in many time-constrained routing and scheduling problems and, thus, deserves some specific attention. The problem consists of finding the least-cost route between any two specified nodes in a network whose nodes can only be visited within a specified time interval. The description of the problem that follows is based on Desrosiers et al. (1995).

Let $G = (\boldsymbol{V}, \boldsymbol{A})$ be a network where $\boldsymbol{A}$ is the set of arcs and $\boldsymbol{V}$ the set of nodes $\boldsymbol{N} \cup \{o, d\}$. $\boldsymbol{N}$ consists of nodes that can be visited from an origin o to a destination d. With each node $i \in \boldsymbol{V}$ a time window $[g_i, h_i]$ is associated. A path in G is defined as a sequence of nodes $i_0, i_1, ..., i_K$ such that each arc (i_{k-1}, i_k) belongs to $\boldsymbol{A}$. All paths start at time g_o from node i_0 and finish at $i_K = d$ no later than h_d. A path is elementary if it contains no cycles. Each arc $(i, j) \in \boldsymbol{A}$ has a positive or negative cost c_{ij} and a positive duration t_{ij}. Service time at node i is included in t_{ij} for all $i \in \boldsymbol{N}$. An arc (i, j) in the set $\boldsymbol{A}$ is defined to be feasible only if it respects the condition: $g_i \le t_i \le h_i$.

The mathematical programming formulation of the shortest-path problem with time windows involves two types of variables: flow variables x_{ij} with $(i, j) \in \boldsymbol{A}$ and time variables t_i with $i \in \boldsymbol{V}$. Using this notation the shortest-path problem with time windows may be formulated as follows (Desrosiers et al. 1995):

$$\min \sum_{(i,j)\in \boldsymbol{A}} c_{ij}\, x_{ij}, \tag{17}$$

subject to

$$\sum_{j\in \boldsymbol{V}} x_{ij} - \sum_{j\in \boldsymbol{V}} x_{ji} = \begin{cases} +1 & i = o, \\ 0 & \text{for } i \in \boldsymbol{N}, \\ -1 & i = d. \end{cases} \tag{18}$$

$$x_{ij} \ge 0 \qquad \text{for } (i,\ j) \in \boldsymbol{A}, \tag{19}$$

$$x_{ij}\,(t_i + t_{ij} - t_j) \le 0 \qquad \text{for } (i,\ j) \in \boldsymbol{A}, \tag{20}$$

$$g_i \le t_i \le h_i \qquad \text{for } i \in \boldsymbol{V}. \tag{21}$$

The objective function (17) attempts to minimise the total travel cost. Constraints (18) and (19) define the flow conditions on the network, while time windows appear in constraint (20). Compatibility requirements between flow and time variables are given in expression (21). This nonlinear problem with time windows is appealing because it can be shown that if the problem is feasible, then there is an optimal integer solution (see Desrosiers et al. 1995).

The problem can be solved by dynamic programming. For the introduction of this approach, define $Q(\boldsymbol{S}, i, t)$ as the minimum cost of the path routing from node o to node i [$i \in \boldsymbol{N} \cup \{d\}$] visiting all nodes in the set $\boldsymbol{S} \subseteq \boldsymbol{N} \cup \{d\}$ only once, and servicing node i at time t or later. The cost $Q(\boldsymbol{S}, i, t)$ can be calculated by solving the following recurrence equations

$$\boldsymbol{Q}(\Phi, o, g_o) = 0\,, \tag{22}$$

$$\begin{aligned} \boldsymbol{Q}(\boldsymbol{S}, j, t) = \min \{ & Q(\boldsymbol{S} - \{j\}, i, t') + c_{ij} \\ & \text{with } i \in \boldsymbol{S} - \{j\},\ t' \le t - t_{ij},\ g_j \le t' \le h_i \} \\ & \text{for all } \boldsymbol{S} \subseteq \boldsymbol{N} \cup \{d\} \text{ for } j \in \boldsymbol{S} \text{ and } g_j \le t \le h_j. \end{aligned} \tag{23}$$

The optimal solution is given by

$$\min_{\boldsymbol{S} \subseteq \boldsymbol{N} \cup \{d\}} \ \min_{g_d \le t \le h_d} Q(\boldsymbol{S}, d, t). \tag{24}$$

It is worthwhile to note that expression (23) is valid only if $g_j \le t \le h_j$. If $t > g_j$, then $Q(\boldsymbol{S}, j, t) = Q(\boldsymbol{S}, j, g_j)$, and if $t > h_j$, then $Q(\boldsymbol{S}, j, t) = \infty$. The shortest-path problem with time windows is NP-hard in the strong sense. Therefore, the dynamic programming algorithm suggested by Desrosiers et al. (1995) has an exponential complexity, and no pseudo-polynomial algorithm is known for this problem.

4 Concluding Remarks

GIS-T, once the sole domain of public sector planning and transportation agencies, is increasingly being used in the private sector to support logistics in general and distribution and production logistics in particular. The cost of the technology is now within the reach of even smaller enterprises. The cost of acquiring the data to populate a GIS for transportation-related applications is falling rapidly. The availability of data is paralleled by GPS services to reference locations accurately. These trends suggest that GIS-T has arrived as a core technology for transportation (Sutton and Gillingham 1997).

The performance of GIS-T software largely depends on how well nodes and links and transportation-related characteristics are arranged into a data structure.

The data structure must not only represent the complexities of transport networks in sufficient detail, but also allow for rapid computation of a wide variety of sophisticated network procedures, such as traveling-salesman problem, vehicle-routing problem, and shortest-path problem algorithms, based on actual network drive time, and not straight-line distances.

References

Assad A.A. and Golden B.L. (1995): Arc routing methods and applications. In: Ball M.O., Magnanti T.L., Monma C.L. and Nemhauser G.L. (eds.) *Handbooks in Operations Research and Management Science, Volume 8*, Elsevier, Amsterdam, pp. 375-483

Atzeni P., Ceri S., Paraboschi S. and Terlone R. (1999): *Database Systems*, McGraw Hill, Berkshire

Bell M. and Iida Y. (1997): *Transportation Network Analysis*, John Wiley, Chichester [UK], New York

Bespalko S.J., Sutton J.C., Wyman M., Veer J.A. van der and Sindt A.D. (1998): Linear referencing systems and three dimensional GIS, Paper presented at the 1998 Annual Meeting of the Transportation Research Board, TRB Paper No. 981404

Bodin L., Golden B.L., Assad A. and Ball M. (1983): Routing and scheduling of vehicles and crews: The state of the art, *Computers & Operations Research*, 10 (20), 63-211

Dane C. and Rizos C. (1998): *Positioning Systems in Intelligent Transportation Systems*, Artech House, Boston

Dantzig G.B., Fulkerson D.R. and Johnson S.M. (1954): Solution of a large-scale traveling-salesman problem, *Operations Research* 7, 58-66

Desrosiers J., Dumas Y., Solomon M.M. and Soumis F. (1995): Time constrained routing and scheduling. In: Ball M.O., Magnanti T.L., Monma C.L. and Nemhauser G.L. (eds.) *Handbooks in Operations Research and Management Science, Volume 8*, Elsevier, Amsterdam, pp. 35-139

Fisher M. (1995): Vehicle routing. In: Ball M.O., Magnanti T.L., Monma C.L. and Nemhauser G.L. (eds.) *Handbooks in Operations Research and Management Science, Volume 8*, Elsevier, Amsterdam, pp. 1-33

Fisher M. and Jaikumar R. (1981): A generalised assignment heuristic for vehicle routing, *Networks* 11, 109-124

Fohl P., Curtin K.M., Goodchild M.F. and Church R.L. (1996): A non-planar, lane-based navigable data model for ITS. In: Kraak M.J. and Molenaar M. (eds.) *Proceedings, Seventh International Symposium on Spatial Data Handling*, Delft, August 12-16, pp. 7B.17-7B.29

Glover F. (1989): Tabu search, part I, *ORSA Journal on Computing* 1 (3), 190-206

Glover F. (1990): Tabu search, part II, *ORSA Journal on Computing* 2 (1), 4-32

Golden B.L. and Assad A.A. (1986): Perspectives on vehicle routing. Exciting new developments, *Operations Research* 14, 803-810

Goodchild M.F. (1998): Geographic information systems and disaggregate transportation modelling, *Geographical Systems* 5, 19-44

Johnson D.S. (1990): Local optimisation and the traveling-salesman problem. In: Goos G. and Hartmanis J. (eds.) *Automata, Languages and Programming*, Springer, Berlin, Heidelberg, New York, pp. 446-461

Kirkpatrick S., Gelatt C.D. Jr. and Vecchi M.P. (1983): Optimization by simulated annealing, *Science* 220, 671-680

Kwan M.-P., Golledge R.G. and Speigle J.M. (1996): A review of object-oriented approaches in geographic information systems for transportation modelling, Draft, Department of Geography, University of California at Santa Barbara

Laporte G. (1992): The traveling-salesman problem: An overview of exact and approximate algorithms, *European Journal of Operational Research* 59 (2), 231-247

Lawler E.L., Lenstra J.K., Rinnoy Kan A.H.G. and Shmoys D.B. (1985): *The traveling-salesman problem: A guided tour of combinatorial optimisation*, John Wiley, Chichester [UK], New York

Lin S. (1965): Computer solutions of the traveling-salesman problem, *Bell System Technical Journal* 44, 2245-2269

Lin S. and Kernighan B. (1973): An effective heuristic algorithm for the traveling-salesman problem, *Operations Research* 21, 498-516

Longley P.A., Goodchild M.F., Maguire D.J. and Rhind D.W. (2001): *Geographic Information Systems and Science*, John Wiley, Chichester [UK], New York

Miller H.J. and Shaw S.-L. (2001): *Geographic Information Systems for Transportation. Principles and Applications*, Oxford University Press, Oxford

Nyerges T.L. (1990): Locational referencing and highway segmentation in a geographic information system, *ITC Journal* 60 (3), 27-31

Potvin J.-Y. (1993): The traveling-salesman problem: A neural network perspective, *ORSA Journal on Computing* 5 (4), 328-348

Spear B.D. and Lakshmanan T.R. (1988): The role of GIS in transportation planning and analysis, *Geographical Systems* 5, 45-58

Sutton J. (1997): Data attribution and network representation issues in GIS and transportation, *Transportation Planning and Technology* 21, 25-44

Sutton J. and Gillingwater D. (1997): Geographic information systems and transportation – Overview, *Transportation Planning and Technology* 21, 1-4

Vonderohe A. and Hepworth T. (1996): A methodology for design of a linear referencing system for surface transportation, Research Report, Sandia National Laboratory, Project AT-4567

5 Expert Systems and Artificial Neural Networks for Spatial Analysis and Modelling:

Essential Components for Knowledge Based Geographical Information Systems

The chapter outlines the general architecture of a knowledge based GISystem that has the potential to intelligently support decision making in a GIS environment. The efficient and effective integration of spatial data, spatial analytic procedures and models, procedural and declarative knowledge is through fuzzy logic, expert systems and neural network technologies. A specific focus of the discussion is on the expert system and neural network components of the system, technologies which had been relatively unknown in the GIS community at the time this chapter was written.

1 Introduction

A geographical information system (GIS) may be defined as a computer-based information system which attempts to capture, store, manipulate and display spatially referenced data (in different points in time), for solving complex research, planning and management problems. The system may be viewed to embody

- a *database of spatially referenced data* consisting of locational and associated attribute data, (large-scale) data sets where the data included usually have special characteristics such as spatial (space-time) dependencies, non-stationarity, varying degrees of reliability, multivariate non-normality, nonlinearities, sensitivity to scale and aggregation effects; noise because of error propagation or due to the nature of data sources (see Openshaw et al. 1990, Openshaw 1992b),

- appropriate *software components* encompassing procedures for the interrelated transactions from input via storage and retrieval, and manipulation and spatial analysis facilities to output, and

- associated *hardware components* including high-resolution graphic display, large-capacity electronic storage devices and processing units

which are organised and interfaced in an efficient and effective manner to allow rapid data storage, retrieval and management capabilities and to facilitate the analysis. Spatial query possibilities and spatial analysis capabilities distinguish GIS from computer-aided cartography (CAC), computer-aided design (CAD) and related systems. Geographic information systems are currently being used for a wide range of government, business and private activites including real time navigation for ship, planes and automobiles; registration and organisation of pipeline networks and other services such as electricity and regional planning (for example, housing and labour market studies); environmental monitoring and management; they are used for inventory, analysis and decision making in all kinds of natural resource studies, agriculture and forestry.

Current geographical information systems have realised only a small portion of their potential up to now. Their lower-level functionality consists of simple statistical graphics, data transformation and computation of basic summary statistics, occassionally including network analysis tools. The lack of integration of analytical and simulation based spatial models is generally perceived as a major shortcoming. Model based analysis provides the central focus for GIS in the social and economic sciences, urban and regional planning, environmental planning and decision making that will enable them to provide the types of information that prospective users in the public and private sphere require (see also Clarke 1990). The development of effective model based geographic information systems is likely to be a key factor in the take up of GIS of the socio-economic fields.

Recent developments in expert systems and neurocomputing concerned with non-programmed adaptive information systems called artificial neural networks have a far reaching potential in this context as building blocks or modules of tomorrow's information systems. Their potential is, however, relatively unknown and unexplored in the GIS community. This paper makes a modest attempt to fill this gap in providing a state-of-the-art review, and hopes to stimulate research in coupling expert systems and artificial neural network components with a GIS environment.

The paper is structured as follows. The system architecture of a knowledge based geographic information system is discussed in Section 2 where rule-based expert systems and artificial neural networks are essential components. Section 3 briefly summarises the expert system component while in sections 4 and 5 major emphasis is laid upon the artificial neural network component. Section 4 describes some basic principles and characteristics of neurocomputing to enable one to understand the application potential of artificial neural networks to be discussed in Section 5.

2 System Architecture of a Knowledge Based Geographic Information System

Current commercially available geographic information systems suffer from three major limitations (see Clarke 1990, Goodchild 1991, Fischer and Nijkamp 1992, Leung 1992):

- The first major deficiency is caused by the *logical foundation*. Geographic information systems are predominantly based on the classical concept of Boolean logic and classical set theory which do not tolerate imprecision in information, human cognition, perception and thought processes. Boolean logic imposes artificial precision on intrinsically imprecise spatial data, phenomena and processes. It is inadequate to handle imprecision of information and knowledge in the representation of spatial data and relationships, in the query and analysis of spatial information. This limitation calls for a more general and sound logical foundation of geographic information systems as offered by the concept of fuzzy logic (see Leung 1992).

- The second major deficiency refers to the *limited built-in analytical and modelling functionality*. Current geographic information systems are strong in the domains of data capture, storage, retrieval and graphical display. Their current capabilities for more sophisticated forms of spatial analysis and modelling, however, are rather limited. There is an enormous range of GIS-relevant spatial procedures and techniques which might be taken into consideration to increase the analytic and modelling functionality of geographic information systems, including inter alia exploratory spatial pattern analysis, regional taxonomic procedures, spatial interaction and choice models, spatial regression models with spatially autocorrelated errors, location-allocation models and space-time statistical models (see Openshaw 1990, Goodchild 1991, Fischer and Nijkamp 1992).

- The third major deficiency of the systems refers to the *low level of intelligence* in terms of knowledge representation and processing. Geographical problems are highly complex in nature. Effective solutions of such problems require an intelligent use of large databases, structured and unstructured knowledge. Over the years, structured (procedural or rule-like) knowledge taking the format of statistical and mathematical models has been developed in Quantitative and Theoretical Geography, and recently loosely coupled with conventional geographic information systems via data export and import (see Leung 1992). The application of new artificial intelligence principles and technologies in general, and expert systems and artificial neural networks in particular, provides the potential to increase the level of intelligence of geographic information systems.

The deficiencies of conventional geographic information systems mentioned above point to three critical elements for integration into the next generation of more intelligent systems:

- the concept of fuzzy logic,
- advanced spatial analysis and models modules via conventional tools and/or via
- artificial intelligence technology in general and spatial expert systems (SES) and artificial neural networks (ANN) in particular.

Figure 1 outlines a system architecture of a knowledge based geographic information system which takes these elements into account and, thus, would greatly enhance the power and usefulness of geographic information systems for spatial analysis and decision making in such a way that GIS can offer intelligent advice or make an intelligent decision about geoprocessing functions. The architecture is based upon three major components:

- the *GIS core component* includes the databases (locational, temporal and attribute data), a database management system(s) and an information retrieval module,

- the *spatial expert system* (SES) *component* for clearly defined and relatively simple spatial analysis and modelling tasks, with a knowledge acquisition module, a (spatial and non-spatial) domain-specific knowledge base and a rule-based inference engine,

- the *artificial neural networks* (ANN) *component* for more sophisticated forms of spatial analysis and modelling.

The SES- and the ANN-components may be integrated with the GIS component through a (fuzzy) information retrieval module. Such an integration establishes communication between expert systems and artificial neural networks on the one side and GIS on the other. The system contains the two major types of knowledge, procedural knowledge (algorithms, mathematical and statistical models) basically via the artificial neural network component and declarative (rule-based) knowledge basically via the expert system component.

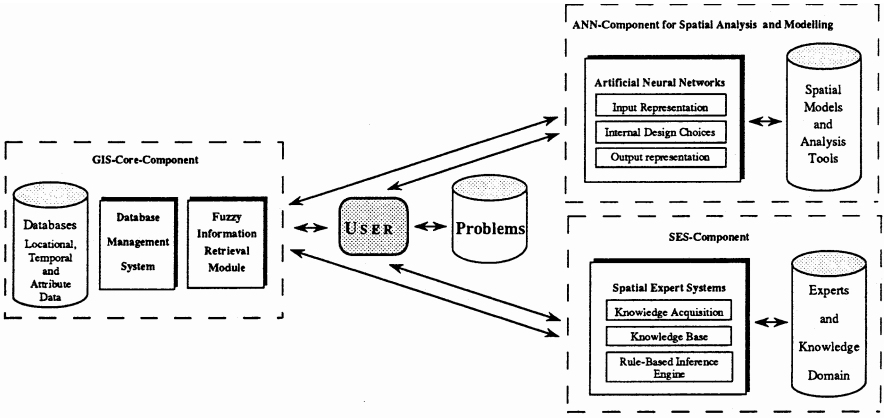


Figure 1 System architecture of a knowledge based geographic information system

To account for imprecision in spatial data as well in the reasoning process fuzzy logic extending the operations of Boolean algebra to cover fractional truth-values intermediate between 0 (false) and 1 (true) may be employed as the logic

foundation in the design of the system (see Leung 1992, Wang et al. 1990). This implies the integration of a fuzzy relational data model, a fuzzy information retrieval tool and fuzzy-logic-based expert systems. Both the artificial neural network and the expert system modules are coupled with the GIS environment. User interfaces may facilitate communication with the GIS environment, the expert systems and spatial models and analysis based artificial neural networks.

This system architecture would enable one to link spatial analysis and modelling with GIS intelligently for specific domains, via the SES- the ANN-components, and would assist the user to choose the best set of procedures and tools to solve his problem at hand within the constraints of data, data quality, cost and accuracy. The neural network and the expert system components fundamentally differ in their knowledge representation techniques. Expert systems utilise symbolic encoding of knowledge in the form of production rules (forward/backward chaining systems), the mainstream approach to knowledge representation, while artificial neural networks commited to the principle of interconnectivity represent knowledge implicitly rather than explicitly.

3 The Expert System Component

Expert systems may be viewed as systems which achieve expert-level performance utilising symbolic representation of knowledge, inference and heuristic search. They are designed to provide acceptable solutions using knowledge from experts and emphasise domain-specific knowledge rather than more general problem solving strategies. Four kinds of software tools are available for developing expert systems:

- conventional languages such as C-language,
- AI languages such as Lisp and Prolog,
- expert system shells (for example, NEXPERT), and
- AI development environments such as KEE (Knowledge Engineering Environment).

Expert system shells and AI development environments greatly reduce the cost of developing expert systems. Spatial expert systems, i.e. expert systems with a spatial domain, have evolved via a manner which parallels that of expert systems in the business data processing community (see, for example, Smith et al. 1987, Kim et al. 1990). A fully fledged fuzzy-logic-based expert system consists of four essential components (see Figure 2):

- a *knowledge acquisition module* to assist in expressing knowledge in a form suitable for inclusion in the knowledge base,

- a *fuzzy knowledge base* consisting of spatial and non-spatial knowledge in fuzzy and non-fuzzy terms [(spatial) objects and their attributes, relationships and their attributes, etc.)] about some substantive domain,

- a *fuzzy-logic-based inference engine* consisting of rule-based inference procedures and control mechanisms used to detect, select, and execute relevant rules in the knowledge base,

- an *user interface* which assists the user to consult the spatial expert system.

In fuzzy-logic based expert systems fuzzy logic is employed to handle appropriate reasoning so that fuzzy and non-fuzzy terms can be employed to make inferences. The *user interface* is a general module which controls I-O-behaviour of the system and facilitates user interaction with the system. The interface fulfills basically two functions: first, to provide the user with the information required to solve his problem, to display the conclusions and to explain its reasoning; second, to translate queries from the user into specific goals for the expert system's engine machine.

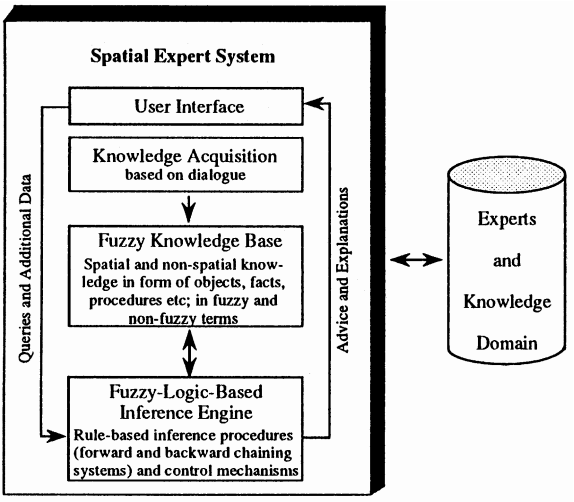


Figure 2 The expert system component of a knowledge based geographic information system

The *knowledge acquisition* module serves to interact with the domain's expert to acquire information relevant to both the knowledge domain and judgemental behaviour of the expert, in terms of objects, facts, fuzzy terms etc. The conventional approach in knowledge acquisition is interview-based [and by static analysis]. Interviewing is essential in eliciting new knowledge from domain experts. The purpose of an interview is to acquire knowledge useful in problem solving. The

fundamental challenge is to decide what kind of knowledge to ask for in what situations. Major recent efforts are directed towards computer-based knowledge acquisition systems which, however, suffer from the so-called knowledge acquisition dilemma (see Kawaguchi et al. 1991). If the system is ignorant, it cannot raise good questions, and if it is sufficiently knowledgeable, it may not raise questions. Consequently, special attention is being paid to identifying what knowledge to give a system in advance and how to use that advanced knowledge to facilitate knowledge acquisition.

The knowledge base contains all the information that is specific to a particular application, and is elicitated from a domain expert and reformulated as a collection of rules or a frame-based structure. Rules derived from experts may be inexact, measurements unreliable, etc. This is especially true in complex spatial tasks and demands for a fuzzy, rather than a precise knowledge base. The production rules characteristically show a format of the general form of *if* [*conditions*] / *then* [*actions*].

The *inference engine* consists of search and reasoning procedures which enable an expert system to arrive at conclusions. The choice of an inference strategy and the choice of knowledge representation are inextricably bound together. In the rule-based systems it is common practice to employ all or some of the following reasoning strategies: backward chaining, forward chaining and/or Bayesian inferences (see Graham 1989). Rule-based reasoning, however, limits a system's ability to acquire knowledge from domain experts. One of the solutions to the knowledge acquisition problem is to reduce dependency on domain experts as much as possible. Several alternatives, such as model-based reasoning, case-based reasoning, and explanation-based learning have been analysed. Model-based reasoning utilises a domain model of structural and functional knowledge about a given target system. It can deal with new situations which rule-based reasoning cannot cover, but its major drawback is the amount of the measuring or testing it must perform to find solutions. Case-based reasoning utilises past problem-solving cases, including success and failure stories which directly reflect domain experts' experience. Finally, explanation-based learning, a deductive learning procedure, is a framework of generating compiled knowledge from goal concepts, training examples, domain theory and operationality criteria (see Kobayashi and Nakamura 1991).

The advantages gained by the integration of an expert system into a GIS world are basically derived from the qualities of the rule-based languages (such as Prolog) (see Webster 1990):

- symbols representing ideas, expectations, adjectives as well as numerical data can be subjected to the same general inferential processing, the semantic flexibility makes rule-based data processing of interest in the search for more semantically-oriented database models,

- the ability to integrate database operations with processing rules implying efficient data processing,

- the ease with which programs and knowledge bases can be amended via ad hoc processing rules,

- the new level of flexibility gained when facing the question of how much information to codify in explicit data representation and how much to leave for deriving via processing rules.

Up to now, however, there are only a few applications of expert systems in spatial analysis and search (see Smith et al. 1987, Kim et al. 1990, Webster 1990, Leung 1992). Compared to other disciplines, research on expert system application has lagged in geography and regional science, due to several reasons. One fundamental reason might be disparities between the type of problems geographers and regional scientists are usually dealing with and the type of problems for which the approach of expert systems is suited. Experience with rule-based expert systems shows that the set of rules required to accomplish multidimensional and complex tasks characteristic to the GIS world is often quite large.

In addition, developing rules and related heuristics may be extremely time consuming and only feasible for relatively simple, well-bounded problem situations in which clear diagnostic rules and procedures are known a priori. The major application domains of GIS in which expert systems can be applied include:

- *automated cartographic procedures* for display (such as name placement and map generalisation),

- *automated device routines* for extracting, sorting, describing data and object structure (for example, a feature extraction detecting valleys and streams using the procedural knowledge in the knowledge base),

- *coupling expert systems and specific spatial analysis and model tools* to provide qualitative reasoning capability (translation of qualitative criteria into numeric input and translation of the output to qualitative concepts) and more intelligent interfaces to the user.

Expert systems may be expected to be used in all conponents of GIS in future, especiallly in terms of object oriented approaches. The major force driving such applications will be the desire to construct more powerful models of complex spatio-temporal phenomena.

4 The Artificial Neural Network Component

Artificial neural networks (neurocomputing) owe their current popularity to two major sources: first, significant major breakthroughs in the design and application of artificial neural networks in the 1980s; second, the new technologies such as

optical processing of information, high-density semiconductor networks, and eventually new materials like the 'spin-glasses' which offer an unforeseen capacity for computation. Improvements in hard- and software have created a climate which has encouraged experimentation and simulation of neural networks.

Artificial neural networks – inspired by models of the human brain and nerve cells – may be viewed as structured networks of highly interconnected processing units or processors (often also termed neurons, in analogy to biological neural networks) with modifiable interconnection weights (Baldi and Hornik 1989). They process information in parallel, carry out simulations almost simultaneously, and work statistically on a trial-and-error basis to solve problems. They use a connectionist type of knowledge representation. Knowledge is not stored in specific memory locations, but related to network structure and consists of the overall state of the network at some equilibrium state. They do not require any instructions, rules and processing requirements about how to process the input data. They learn by being shown examples and the expected results and have the ability to respond to input which they have not seen before (i.e. to generalise), where input may be partial or incomplete. But when neural networks generalise, they may be wrong, and mistakes may be hard to undo or forget. Errors are spread out over the connections. It is hard to know which processing units to blame.

Formally, an ANN may be viewed as a dynamic system with the topology of a directed graph (see Hecht-Nielsen 1990). The nodes of the graph are called processing elements (PEs), and the directed links (unidirectional signal channels) are termed connections (occasionally synapses). The processing elements communicate with one another by signals that are numerical rather than symbolic. An ANN is termed feedforward or feedback according to whether the directed graph is acyclic or cyclic. ANNs are designed to perform a task (such as, for example, spatial modelling) by specifying the number of processing elements, the network topology (i.e. the interconnections of the processing elements), and the weight or strength of each connection via learning rules.

Typically, the processing elements are organised into a hierarchical series of levels (layers): an input layer, one or more intermediate (so-called hidden) layer(s), and an output layer. Figure 3 shows a neural network with three layers of processing units, a typical organisation for the popular neural network architecture which is known as multi-layered feedforward. The processing elements of the input layer assume the values of an input pattern represented as a vector which is input to the network. The intermediate or hidden layer consists of processing elements which receive and transmit the input signals, and send copies of their output signals to other processors via output connections and the outside world. Each of the copies carries the same identical signal of the processor. Sometimes, there is more than one intermediate layer. These intermediate layer processors are connected to output units which form the output layer making the network information available to the external environment. There are no direct connections from input to output that skip the hidden layer. Each interconnection between processing elements acts as a communication route. Numeric values are passed along these interconnections from one processor to another. They are weighted by connection strengths which are associated with each interconnection and adjusted

during training to generate the final neural network. The processing performed by the network in transforming the input pattern depends on the set of connections. These weights are viewed as encoding the system's knowledge.

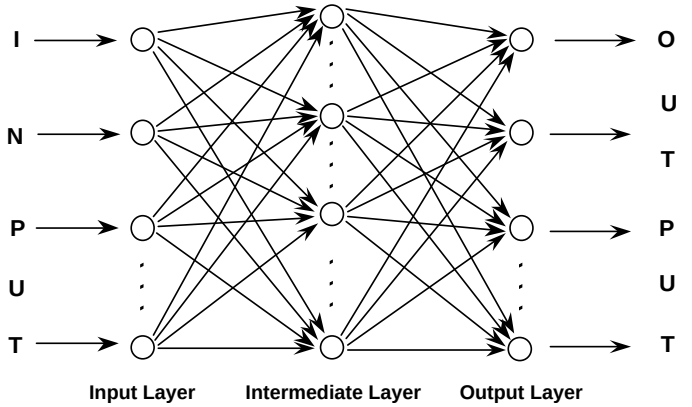


Figure 3 A feedforward neural network with three fully interconnected layers

Figure 4 depicts an example of a *typical processing unit* i in layer $k+1$ ($k \geq 1$) of a multi-layer ed network. Except for the input layer processors, the net input to each processor is the sum of the weighted outputs of the processors from the layer below. Therefore, for a processor i in layer $k+1$ ($k \geq 1$), the net input is

$$net_i^{k+1} = \sum_j \mu_{ij}^k \; o_j^k \tag{1}$$

where μ_{ij}^k are the connection weights between layers k and $k+1$, and o_j^k the output of processor j from layer k. The output o_i of each (continuous-valued) processor i in an intermediate layer $(k+1)$ is a continuous function (so-called activation function) of its net input net_i

$$o_j^{k+1} = f(net_i^{k+1}) = f\left(\sum_j \mu_{ij}^k \; o_j^k\right). \tag{2}$$

The nonlinearities are located in the activation (nonlinear transfer) function $f(\text{net}_i)$ of the hidden units. In most cases f has a saturation nonlinearity so that f levels off and approaches fixed limits for large negative and positive nets. Then o_j^{k+1} will always remain between those limits. One particular functional structure which is often used is the sigmoid shaped function for a (0,1) range

$$o_j^{k+1} = f(net_i^{k+1}) = \frac{1}{1+\exp-(-\beta\, net_i^{k+1})} \tag{3}$$

where β is an adjustable gain parameter which controls the steepness of the output transition. Typically, $\beta = 1$, and (3) is often referred to in the literature as a logistic function. An interesting observation is that the average firing frequency of biological neurons, as a function of excitation, follows a sigmoidal characteristic.

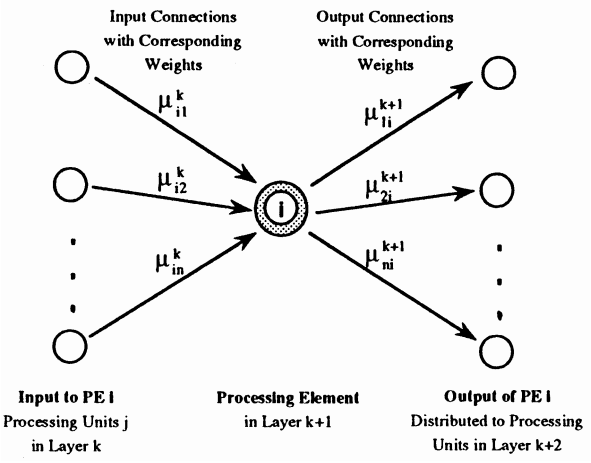


Figure 4 A typical processing unit from a layered artificial neural network

ANN learning always takes place in accordance with a training regimen. That is, the ANN is subjected to a particular information environment. At the most fundamental level, training regimens can be divided into two major categories (see Hecht-Nielsen 1990):

Supervised training implies a situation in which the network is supplied with a sequence (x_k, y_k) of input data x_k and desired or target data y_k, and the network is thus told precisely what should be emitted as output. For ANNs which use supervised training, the challenge is to develop a learning law or rule that will efficiently guide the vector of adaptive weights to a location which minimises or maximises some particular global neural network performance criterion such as mean squared error. An example of supervised learning in a feedforward network is the generalised delta rule – also known as error backpropagation algorithm – which updates the weights by the method of steepest descent (Rumelhart et al. 1986), and an example of supervised learning in a recurrent (feedback) structure is the Hopfield CAM (content addressable memory) approach (Hopfield 1981).

Unsupervised training, sometimes called self-supervised training, is a biologically more plausible model of learning. It is a training regimen in which the network is given only input data. The network uses no external influences to adjust its weights, but local information about neuronal signals. There are no *y*-inputs. Instead there is an internal monitoring of performance. Basically, the network looks for regularities or trends in the input signals and produces output vectors that are consistent. The training process extracts the statistical properties of

training sets and groups similar vectors into classes. An unsupervised learning algorithm might emphasise cooperation among clusters of processing elements.

Competition between processing elements could also form the basis of learning (see Nelson and Illingworth 1990). An example of an unsupervised learning in a feedforward network is the Kohonen self-organising network (Kohonen 1984), whereas the ART (adaptive resonance theory) approach exemplifies unsupervised learning with a recurrent network structure (Carpenter and Grossberg 1987). At the present state of the art, unsupervised learning is not well understood and is still subject of much research. It is of great interest in many high-speed real time GIS environments where there is not enough time, information, or computational precision to use supervised algorithms.

Neural networks are proving to be a viable technology. But they are no more a panacea than are other computing techniques. They have their own peculiar problems and drawbacks: a steep learning curve, the danger of overfitting, a lack of standard development methodologies, demanding preprocessing requirements, and integration issues. All these problems need to be addressed, and will be in time (Hall 1992).

5 The Potential Role of Neural Networks in Geographic Information Processing

Neural networks have a far-reaching potential as modules in tomorrow's computational world in general and in knowledge based geographic information systems in particular. Useful applications have been already designed, built and comercialised in various fields, such as

- *image analysis,* i.e. pattern classification and pattern completion problems, in various domain areas (for example, automated medical image analysis, industrial visual inspection of a product or component under manufacture),

- *automated diagnosis* ranging from machine diagnosis and failure analysis to identify and evaluate fault types (for example, jet engine and automobile engine diagnosis) to automated control covering a wide range of complexity of control problems, from simple systems such as balancing a broom to complex systems such as autonomous control of a moving car and robotic control problems,

- *speech analysis and generation, including text-to-speech* translation and automated speed (syllable) recognition where current applications, however, are limited to the recognition of phenomes or simple words and a limited vocabulary.

Up to now, geographers and regional scientists have been rather slow in realising the great potential of the revolutionary new technology of neural networks, with

the exception of very few scholars like White (1989), Halmari and Lundberg (1991), Fischer and Gopel (1992) and Openshaw (1992a, b).

ANNs have been recognised as being unusually well suited for solving pattern recognition problems. Current examples are digital image processing and analysis, automated character recognition, recognition and understanding of speech. In fact many pattern recognition applications arise in situations where they are not expected. Examples of these are weather forecasting and stock market analysis. Broadly speaking, pattern recognition concerns the description or classification (recognition) of measurements. Although there are some exceptions, pattern recognition problems tend to involve one of two different classes of data: images and space-time series.

Of the application areas to which neurocomputing may be applied in a GIS context, those in the area of spatial data analysis seem to be among those having the highest potential. The term spatial analysis in a broader sense is usually taken to mean the development of a predictive model or other summary representation of a large body of spatially referenced data. The range of potential applications is impressive. Key candidate application areas in geographic information processing are summarised in Figure 5 opposite. They include

- *exploratory spatial data and image analysis*, i.e. pattern detection and pattern completion via supervised or unsupervised neural network architectures, especially in the field of environmental monitoring and management in remote sensing and data-rich GIS environments,

- *homogeneous and functional regional taxonomic problems,* especially in the case of very large data sets (see Openshaw et al. 1991 for evaluating different unsupervised neural network classifiers on census data for Britain),

- *spatial interaction and choice modelling* via supervised neural network architectures (see Fischer and Gopal 1992 for the application of multi-layered feed-forward networks with a backpropagation learning algorithm to model telephone traffic in Austria, or Openshaw 1992a to model journey to work flows),

- *optimisation problems* such as the classical traveling-salesman problem and shortest-path problems in networks via supervised neural network architectures, (see Wilson and Pawley 1988 for a Hopfield network application to the traveling-salesman problem), and

- *space-time statistical modelling* via supervised or unsupervised neural networks, depending upon the problem under study.

This list of problems addressable by the neural network approach is by no means exhaustive, but certainly reflects priorities for neural network applications in geographic information processing. In principle, neural networks may be developed to replicate the descriptive and predictive functions of current statistical and

mathematical procedures of any complexity, often with an improved level of performance and accuracy (Openshaw 1992a). As with almost all new technologies, neurocomputing is just one new tool in an already well-equipped toolbox. If neurocomputing is to realise its potential, it will be necessary to blend neurocomputing techniques with other existing techniques.

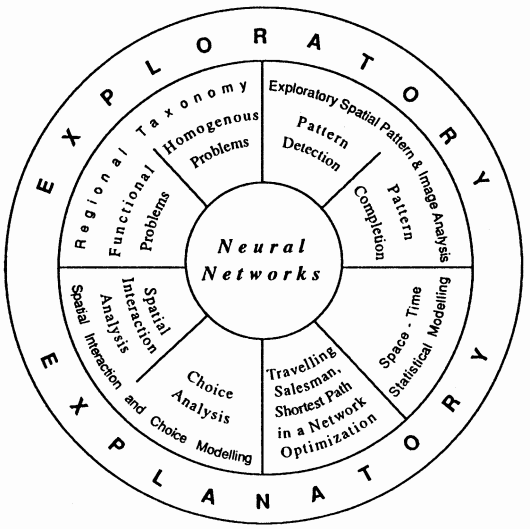


Figure 5 Some key candidate application areas of neural networks in the field of geographic information processing

6 Summary and Conclusions

Knowledge based geographic information systems will play an important role in decision and policy analysis (resource exploration, environmental monitoring and management, land use planning, motor vehicle navigation, distribution logistics, etc.). Geographic information systems without intelligence have few chances to provide effective and efficient solutions to spatial decision making problems in the highly complex and imprecise decision making environment. The application of applied artificial intelligence techniques and principles to geographic information and analysis provides a great potential to meet the challenges encountered in developing the next generation of intelligent GIS. Spatial expert systems and artificial neural networks may be considered to be essential components of such systems. They differ from each other fundamentally in their knowledge representation techniques. Expert systems utilise the way of symbolic encoding of knowledge in form of production rules (forward/backward chaining systems), while artificial neural networks commited to the principle of interconnectivity represent

knowledge implicitly rather than explicitly. Neurocomputing shows greater flexibility than expert systems to deal with situations typical for the GIS world, in which the data at hand are poor (incomplete, noisy, imprecise, etc.) from an analytical point of view. The detection of patterns and relationships in data-rich environments is important, but relevant theories and hypotheses for data analysis are missing.

The future might bring shared geoprocessing with expert systems and artificial neural networks, genetic algorithms and fuzzy logic, especially when geographical information systems are implemented on large multi-processor systems.

References

Baldi P. and Hornik K. (1989): Neural networks and principal component analysis: Learning from examples without locall minima, *Neural Networks* 2, pp. 53-58.

Carpenter G.A. and Grossberg S. (1987): ART 2: Self-organisation of stable category recognition codes for analog input patterns, *Applied Optics* 26 (3), pp. 4919-4930

Clarke M. (1990): Geographical information systems and model based analysis: Towards effective decision support systems. In: Scholten H.J. and Stillwell J.C.M. (eds.) *Geographical Information Systems for Urban and Regional Planning*, Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 165-175

Fischer M.M. and Gopal S. (1992): Neural networks models and interregional telephon traffic: Comparative performance comparisons between multiplayer feedforward networks and the conventional spatial interaction model, Paper presented at the Symposium of the IGU-Commission on Mathematical Modelling, Princeton, August 5-7, 1992 (=WSG-DP 27, Department of Economic and Social Geography, Vienna University of Economics and Business Administration)

Fischer M.M. and Nijkamp P. (eds.) (1992): *Geographic Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York

Goodchild M.F. (1991): Progress on the GIS research agenda. In: Harts J., Ottens H.F.L. and Scholten H.J. (eds.) *EGIS '91. Proceedings, Second European Conference on Geographical Information Systems, Volume 1*, Utrecht: EGIS Foundation, pp. 342-350

Graham I. (1989): Inside the inference engine. In: Forsyth R. (ed.) *Expert Systems, Principles and Case Studies*, Chapman and Hall, London, pp. 57-83

Hall C. (1992): Neural net technology: Ready for prime time? *IEEE Expert* 7 (6), 2-4

Halmari P.M. and Lundberg C.G. (1991): Bridging inter- and intra-corporate information flows with neural networks, Paper presented at the Annual Meeting of the Association of American Geographers, Miami, April 13-17, 1991

Hecht-Nielsen R. (1990): *Neurocomputing*, Addison-Wesley, Reading [MA]

Hopfield J.J. (1984): Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Sciences* 81 (Biophysics), 3088-3092

Kawaguchi A., Motoda H. and Mizoguchi R. (1991): Interview-based knowledge acquisition using dynamic analysis, *IEEE Expert* 8 (5), 47-60.

Kim T.J., Wiggins L.L. and Wright J.R. (eds.) (1990): *Expert Systems: Applications to Urban and Regional Planning*, Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 191-201

Kobayashi S. and Nakamura K. (1991): Knowledge compilation and refinement for fault diagnosis, *IEEE Expert* 8 (5), 39-46

Kohonen T. (1984): *Self-Organisation and Associative Memory*, Springer, Berlin, Heidelberg, New York

Leung Y. (1992): Towards the development of an intelligent decision support system. In: Fischer M.M. and Nijkamp P. (eds.) *Geographical Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 131-145

Mc'Lord Nelson M. and Illingworth W.T. (1990): *A Practical Guide to Neural Nets*, Addison-Wesley, Reading [MA]

Openshaw S. (1992a): Modelling spatial interaction using a neural net. In: Fischer M.M. and Nijkamp P. (eds.) *Geographical Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 147-164

Openshaw S. (1992b): Some suggestions concerning the development of artificial intelligence tools for spatial modelling and analysis in GIS. In: Fischer M.M. and Nijkamp P. (eds.) *Geographical Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 17-33

Openshaw S. (1990): A spatial analysis research strategy for the Regional Research Laboratory Initiative, Regional Research Laboratory Initiative Discussion Paper No. 3

Openshaw S., Cross A. and Charlton M. (1990): Building a prototype Geographical Correlates Exploration Machine, *International Journal of Geographical Information Systems* 4, 297-311

Openshaw S., Wymer C. and Charlton M. (1991): An evaluation of three neural net classifiers on census data for Britain, Paper presented at the 7th European Colloquium on Quantitative and Theoretical Geography, Hasseludden (Sweden), September 5-8, 1991

Rumelhart D.E., Hinton G.E. and Williams R.J. (1986): Learning representations by backpropagating errors, *Nature* 323, 533-536

Smith T.R., Peuquet D., Menon S. and Agarwal P. (1987): KBGIS-II. A knowledge based geographical information system, *International Journal of Geographic Information Systems* 1, 149-172

Wang F., Hall G.B. and Subaryono (1990): Fuzzy information representation and processing in conventional GIS software: Database design and application, *International Journal of Geographical Information Systems* 4, 261-283

Webster C. (1990): Rule-based spatial search, *International Journal of Geographical Information Systems* 4, 241-259

White R.W. (1989): The artificial intelligence of urban dynamics: Neural net modelling of urban structure, *Papers of the Regional Science Association* 67, 43-53

Wilson G.V. and Pawley G.S. (1988): On the stability of the traveling-salesman problem algorithm of Hopfield and Tank, *Biological Cybernetics* 58, 63-70

Part II

Computational Intelligence in Spatial Data Analysis

6 Computational Neural Networks – Tools for Spatial Data Analysis

This chapter is a tutorial text that gives an introductory exposure to computational neural networks for students and professional researchers in spatial data analysis. The text covers a wide range of topics important for developing neural networks into an advanced spatial analytic tool for non-parametric modelling. The topics covered include a definition of computational neural networks in mathematical terms and a careful and detailed description of neural networks in terms of the properties of the processing elements, the network topology and learning in the network. The chapter presents four important families of neural networks that are especially attractive for solving real world spatial analysis problems: backpropagation networks, radial basis function networks, supervised and unsupervised ART models, and self-organising feature map networks.

1 Introduction

The proliferation and dissemination of digital spatial databases, coupled with the ever wider use of Geographic Information Systems (GIS) and Remote Sensing (RS) data, is stimulating increasing interest in spatial analysis from outside the spatial sciences. The recognition of the spatial dimension in social science research sometimes yields different and more meaningful results than analysis that ignores it.

Spatial analysis, as it has evolved over the last few decades, basically includes two major fields: spatial data analysis and spatial modelling, though the boundary is rather blurred. Spatial data analysis may be defined as a core technology [i.e. body of methods and techniques] for analysing events (objects) where the results of analysis depend on the spatial arrangement of the events (see Haining 1994). These may be represented in the form of a point, line or area, in the sense of spatial primitives located in geographical space, to which a set of one or more attributes are attached. Location, topology, spatial arrangement, distance and spatial interaction have become a major focus of attention in activities concerned with detecting patterns in spatial data, and with exploring and modelling the relationships between such patterns.

Empirical studies in the spatial sciences routinely employ data for which locational attributes are an important source of information. Such data characteristically consist of one or more cross-sections of observations for either micro-units such as individuals (households, firms) at specific points in space, or aggregate

spatial entities such as census tracts, electoral districts, regions, provinces, or even countries. Observations such as these, for which the absolute location and/or relative positioning (spatial arrangement) is explicitly taken into account, are termed *spatial data.* In the socioeconomic realm points, lines, and areal units are the fundamental entities for representing spatial phenomena. This form of spatial referencing is also a salient feature of GIS and *Spatial Data Infrastructures*. Three broad classes of spatial data can be distinguished:

- *object data* where the objects are either *points* (spatial point patterns or locational data, i.e. point locations at which events of interest have occurred) or *areas* (area or lattice data, defined as discrete variations of attributes over space),

- *field data* (also termed geostatistical or spatially continuous data), which consists of observations associated with a continuous variation over space, or given values at fixed sampling points, and

- *spatial interaction data* (sometimes called link or flow data) consisting of measurements each of which is associated with a link or pair of locations representing points or areas.

Analysing and modelling spatial data present a series of problems. Solutions to many of them are obvious, others require extraordinary efforts. Data can exercise a considerable power, and misinterpreted they may lead to meaningless results; therein lies the tyranny of data. The validity of quantitative analysis depends on *data quality*. Good data are reliable, contain few or no mistakes, and can be used with confidence. Unfortunately, nearly all spatial data are flawed to some degree. Errors may arise in measuring both the location and attribute properties, but may also be associated with computerised processes responsible for storing, retrieving, and manipulating the data. The solution to the quality problem is to take the necessary steps to avoid faulty data determining research results.

The particular form (i.e. size, shape and configuration) of the spatial aggregates can affect the results of the analysis to a varying, but usually unknown degree (see, e.g., Openshaw and Taylor 1979, Baumann et al. 1983). This problem has become generally recognised as the *modifiable areal unit problem* (MAUP), the term stemming from the fact that areal units are not 'natural' but usually arbitrary constructs. For reasons of confidentiality, social science data (e.g. census data) are not often released for the primary units of observation (individuals), but only for a set of rather arbitrary areal aggregations (enumeration districts or census tracts). The problem arises whenever area data are analysed or modelled, and can lead to two effects: one derives from selecting different areal boundaries while holding the overall size and the number of areal units constant (*the zoning effect*). The other derives from reducing the number but increasing the size of the areal units (*the scale effect*). There is no analytical solution to the MAUP, but the following types of question have to be considered in constructing an areal system for analysis: What are the basic spatial entities for defining areas? What theory guides the

choice of the spatial scale? Should the definition process follow strictly statistical criteria and merge basic spatial entities to form larger areas using some regionalisation algorithms (see Wise et al. 1996)? These questions pose daunting challenges (Fischer 2001).

Most of the spatial data analysis techniques and methods currently in use were developed in the 1960s and 1970s, i.e. in an era of scarce computing power and small data sets. Their current implementation takes only limited advantage of the data storage and retrieval capabilities of modern computational techniques, and basically ignores both the emerging new era of parallel supercomputing and computational intelligence techniques.

There is no doubt that spatial analysis in general and spatial data analysis in particular is currently entering a period of rapid change. It promises to be a period which presents a unique opportunity for developing novel styles of data analysis which will respond to the need to efficiently and comprehensively explore large databases for patterns and relationships, against a background of data uncertainty and noise, especially when the underlying database is of the order of gigabytes. In this chapter, we hope to provide a systematic introduction to computational neural networks (CNNs) with the aim of helping spatial analysts learn about this exciting new field. The hope is that this contribution will help to demystify and popularise computational neural networks in spatial analysis, and stimulate neural network applications in spatial analysis which are methodologically and technically sound, rather than ad hoc. Some of the material in this chapter is similar to introductions to certain computational neural networking in spatial analysis that have already appeared (Fischer 1995, 1998b). Some parts have also been adapted from Fischer and Abrahart (2000).

The chapter is structured as follows. In Section 2 a brief summary is given of the computational appeal of neural networks for solving some fundamental spatial analysis problems. This is followed by a definition of computational neural network models in mathematical terms (Section 3). The next three sections are devoted to a discussion of the three fundamental components of a computational neural network: the properties of the processing elements (Section 4), network topology (Section 5) and learning (determination of the weights on a connection) in a computational neural network (Section 6). In Section 7 a taxonomy of CNNs is presented, breaking neural networks down according to the topology and type of interconnections (network topology) and the learning (training) paradigm adopted. The concluding section pays attention to the question of how neurocomputing techniques differ from conventional statistics.

2 Why Computational Neural Networks?

Briefly stated, a CNN model is a parallel, distributed information structure consisting of a set of adaptive processing [computational] elements and a set of unidirectional data connections (a more detailed definition is given below). These networks are neural in the sense that they have been inspired by neuroscience, but

not necessarily because they are faithful models of biological neural or cognitive phenomena. In fact, the networks covered in this contribution are more closely related to traditional mathematical and/or statistical models such as non-parametric pattern classifiers, statistical regression models and clustering algorithms than they are to neurobiological models.

The notion computational neural networks is used to emphasise rather than to ignore the difference between computational and artificial intelligence. Neglect of this difference might lead to confusion, misunderstanding and misuse of neural network models in spatial analysis. Computational intelligence (CI) denotes the lowest-level forms of 'intelligence' which stem from the ability to process numerical (low-level) data without using knowledge in the artificial intelligence (AI) sense. An AI system (exemplified in a spatial analysis context by spatial expert systems: see Smith et al. 1987, Kim et al. 1990, Webster 1990, Leung 1993) is a CI system where added value comes from incorporating knowledge in form of non-numerical information, rules and constraints that humans process. Thus, the neural networks considered in this chapter, such as feedforward and pattern classifiers and function approximators, are computational rather than AI systems.

Why have neural network based spatial analysis been receiving increasing attention in the last few years? There are a number of reasons. From a computational point of view, they offer four primary attractions which distinguish them qualitatively from the standard information approaches currently in use in spatial analysis.

The strongest appeal of CNNs is their suitability for *machine learning* (i.e. computational adaptivity). Machine learning in CNNs consists of adjusting connection weights to improve the performance of a CNN. This is a very simple and pleasing formulation of the learning problem. Certainly *speed of computation* is another key attraction. In traditional single processor von Neuman computers, the speed is limited by the propagation delay of the transistors. Computational neural networks, on the other hand, because of their intrinsic massively parallel distributed structure, can perform computations at a much higher rate, especially when these CNNs are implemented on parallel digital computers, or, ultimately, when implemented in customised hardware. This feature makes it possible to use CNNs as tools for real time applications involving, for example, pattern recognition in GIS and RS data environments. It is clear that the increasing availability of parallel hardware will enhance the attractiveness of CNNs and other basically parallel models in spatial analysis.

Furthermore, because of their nonlinear nature, CNNs can perform functional approximation and pattern classification operations, which are beyond optimal linear techniques. CNNs offer a *greater representational flexibility* and *freedom from linear model design*. They are semi- or non-parametric and make weaker assumptions concerning the shape of underlying distributions than conventional statistical models. Other important features include *robust behaviour* with noisy data. Noise refers to the probabilistic introduction of errors into data. This is an important aspect of real world applications, and CNNs can be especially good at handling noise in a reasonable manner.

CNNs may be applied successfully in a variety of diverse areas in spatial analysis to solve the problems of the following type.

2.1 Pattern Classification

The task of pattern classification is to assign an input pattern represented by a feature vector to one of several prespecified classes. Well-known applications include spectral pattern recognition utilising pixel-by-pixel spectral information to classify pixels (resolution cells) from multispectral imagery to a priori given land cover categories. As the complexity of the data grows (use of more spectral bands of satellite scanners, and finer pixel and grey level resolutions), so too does the need for more powerful pattern classification tools.

2.2 Clustering or Categorisation

In clustering, also known as unsupervised pattern classification, there are no training data with known class labels. A clustering algorithm explores the similarity between the patterns and places similarity between patterns in a cluster. Well-known clustering applications include data mining, data compression, and exploratory spatial data analysis. Very large scaled pattern classification tasks based on census small area statistics for consumer behaviour discrimination, for example, have proven to be difficult in unconstrained settings for conventional clustering algorithmic approaches, even using very powerful computers (see Openshaw 1995).

2.3 Function Approximation

Suppose a set of n labelled training patterns (input-output pairs), $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ have been generated from an unknown function $\Phi(\boldsymbol{x})$ [subject to noise]. The task of function approximation is to find an estimate, say Φ^*, of the unknown function Φ. Various spatial analysis problems require function approximation. Prominent examples are spatial regression and spatial interaction modelling.

2.4 Prediction/Forecasting

Given a set of n samples $\{y(t_1), y(t_2), ..., y(t_n)\}$ in a time sequence $t_1, t_2, ..., t_n$, the task is to predict the sample $y(t_n + 1)$ at some future time $t_n + 1$. Prediction/forecasting has a significant impact on decision-making in regional development and policy.

2.5 Optimisation

A wide variety of problems in spatial analysis can be posed as (nonlinear) spatial optimisation problems. The goal of an optimisation algorithm is to find a solution satisfying a set of constraints such that an objective function is maximised or minimised. The traveling-salesman problem, an NP-complete problem, and the problem of finding optimal locations are classical examples.

3 Definition of a Computational Neural Network

Computational neural networks are fundamentally simple (generally nonlinear) adaptive information structures. However, it is helpful to have a general definition first before characterising the building blocks of these structures in more detail. In mathematical terms a computational neural network model is defined as a directed graph with the following properties:

- a state level u_i is associated with each node *i*,
- a real-valued weight w_{ij} is associated with each edge *ji* between two nodes *j* and *i* that specifies the strength of this link,
- a real-valued bias θ_i is associated with each node *i*,
- a (usually nonlinear) transfer function $\varphi_i[u_i, w_{ij}, q_i, (i \neq j)]$ is defined for each node *i*. This determines the state of the node as a function of its bias, the weights of its incoming links, and the states of the nodes *j* connected to it by these links.

In the standard terminology, the nodes are called *processing elements* (PEs), processing, computational units or neurons. The edges of the network are called connections. They function as unidirectional conduction paths (signal or data flows) that transmit information in a predetermined direction. Each PE can receive any number of incoming connections called *input connections*. This property is referred to as an unlimited fan-in characteristic of the processing elements, i.e., the number of incoming connections into a node is not restricted by some upper bound. Each processing element can have any number of output connections, but carries an identical processing element state level (also called activity level, activation or output signal). The weights are termed connection parameters and determine the behaviour of the CNN model.

For illustration purposes, a typical CNN architecture is pictured in Figure 1. In the figure the circles denote the processing elements, and the arrows the direction of the signal flow. The single output signal of the processing elements branches into copies which are distributed to other processing elements or which leave the network altogether. The network input represented by the external world can be viewed as data array $\boldsymbol{x} = (x_1, ..., x_n) \in \Re^n$ (*n*-dimensional euclidean space) and the output of the network as data array $\boldsymbol{y} = (y_1, ..., y_m) \in \Re^m$ (*m*-dimensional

euclidean space with $m < n$). When viewed in this fashion, the CNN can be thought of as a function $\Phi: \mathcal{R}^n \rightarrow \mathcal{R}^m$.

The processing elements implement transfer functions (i.e. primitive functions) which are combined to produce Φ, the so-called network function. This observation is the basis for the mechanism to embed CNNs into a programmed information system. Computational neural network models are specified by

- the node characteristics (i.e. properties of the processing elements),
- the network topology (i.e. pattern of connections between the processing elements, also termed network architecture), and
- the methods of determining the weights on the connections (called learning rules or algorithms, machine learning, network training or parameter estimation).

Both design procedures involving the specification of node characteristics and the network topology, and learning rules are the topic of much current research.

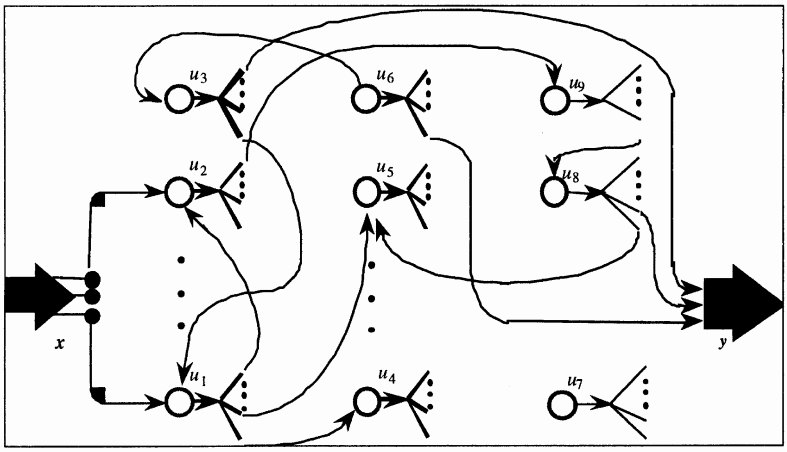


Figure 1 A typical computational neural network architecture

4 Properties of the Processing Elements

Processing elements are fundamental to the operation of any CNN. Thus, it is important to have a closer look at the operation of such elements since they form the basis of the CNNs considered in this chapter. Figure 2 shows internal details of a generic processing unit *i* from an arbitrary computational neural network. The processing unit in this figure occupies a position in its network that is quite general; i.e. this processing element accepts inputs from other processing elements and sends its output (activation) to other processing elements. Processing

elements that are neither input nor output units maintain this generality and function as fundamental nonlinear computing devices in a CNN.

In order to simplify notation, we use u_i to refer to both the processing unit and the numerical activation [output] of that unit where there is no risk of confusion. Each unit u_i computes a single [numerical] unit output or activation. For example, output u_8 in Figure 1 is both an output of the network as a whole and an input for unit u_5 to be used in the computation of u_i's activation. Inputs and outputs of the processing elements may be discrete, usually taking values {0, 1} or {–1, 0, 1}, or they may be continuous, generally assuming values [0, 1] or [–1, +1]. The CNNs we consider in this chapter are characterised by continuous inputs and outputs.

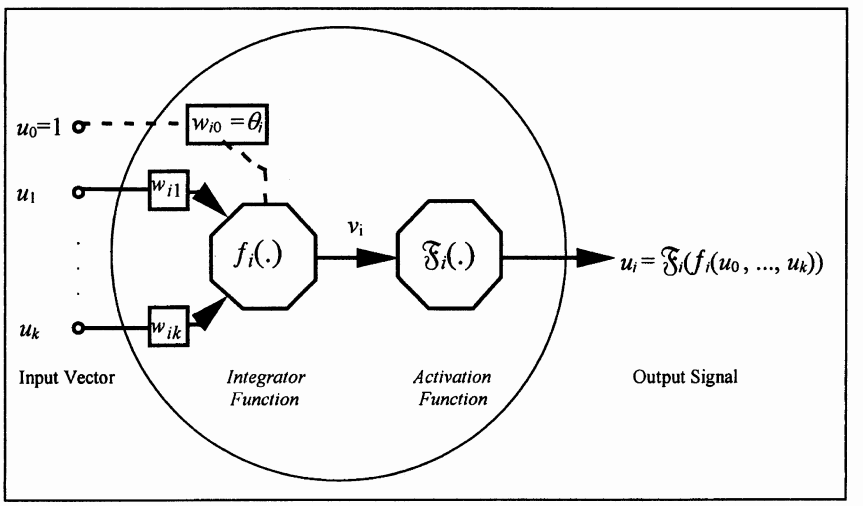


Figure 2 Generic processing element u_i from an arbitrary computational network

Figure 2 assumes that the processing unit in question, u_i, gets k input signals, $\boldsymbol{u} = \{u_1, ..., u_k\}$, arriving via the incoming connections which impinge on unit u_i. Note that the k units are indexed 1 through k so that $k < i$. The corresponding connection weights are w_{ij} $(j = 1, ..., k)$. It is important to make a note of the manner in which the subscript of the connection weight w_{ij} is written. The first subscript refers to the processing element in question and the second subscript refers to the input end of the connection to which the weight refers. The reverse of this notation is also used in the neural network literature. We refer to the weights $\boldsymbol{w}_{i\bullet} = \{w_{i1}, ..., w_{ik}\}$ as the weights of unit u_i. To simplify notation $\boldsymbol{w}$ is used for the vector $\boldsymbol{w}_{i\bullet}$.

Positive weights indicate reinforcement, negative weights represent inhibition. By convention there is a unit u_0, whose output is always +1, connected to the processing element in question. The corresponding weight w_{i0} is called bias θ_i of

unit i (i.e. the bias is treated as any other weight). Thus, we may define the $(k + 1)$-by-1 input vector

$$u = (1, u_1, u_2, ..., u_k)^T \tag{1}$$

and, correspondingly, we define $(k + 1)$-by-1 weight (also called connection weight, connection parameter or simply parameter) vector

$$w = (\theta, w_{i1}, ..., w_{ik})^T. \tag{2}$$

The basic operation of a processing element in computing its activation or output signal u_i involves applying a transfer function φ_i that is composed of two mathematical functions (Fischer 1995): an integrator function f_i and an activation or output function $\mathcal{F}_i$ i.e.

$$u_i = \varphi_i(\boldsymbol{u}) = \mathcal{F}_i (f_i (\boldsymbol{u})) \tag{3}$$

Typically, the same transfer function is used for all processing units in any particular layer of a computational neural network, although this is not essential.

The integrator function f_i performs the task of integrating the activation of the units directly connected to the processing element in question and the corresponding weights for those connections, thus reducing the k arguments to a single value (called net input to or activity potential of the processing element) v_i. Generally, f_i is specified as the inner product of the vectors $\boldsymbol{u}$ and $\boldsymbol{w}$, as follows

$$v_i = f_i(\boldsymbol{u}) = <\boldsymbol{u}, \boldsymbol{w}> = \sum_{j=0,1,...,k} w_{ij} u_j \tag{4}$$

where $\boldsymbol{w}$ has to be predefined or learned during training. In this case the net input to the processing element is simply the weighted sum of the separate outputs from each of the k connected units plus a bias term w_{i0}. Because of the weighted process used to compute v_i, we automatically get a degree of tolerance for noise and missing data (see Gallant 1993). The bias term represents the offset from the origin of the k-dimensional Euclidean space $\mathcal{R}^k$ to the hyperplane normal to $\boldsymbol{w}$ defined by f_i. This arrangement is called a first order processing element because f_i is an affine (linear when $w_{i0} = 0$) function of its input vector $\boldsymbol{u} = (u_1, ..., u_k)^T$. Higher order processing elements arise when more complicated functions are used for specifying f_i. A second order processing unit is, for example, realised if f_i is specified as a quadratic form, say $\boldsymbol{u}^T \boldsymbol{w} \boldsymbol{u}$, in $\boldsymbol{u}$.

The activation or output function, denoted by $\mathcal{F}_i (f_i (\cdot))$, defines the output of the processing unit in terms of the net input v_i at its input. There are various ways of specifying $\mathcal{F}_i$, but usually it is specified as a nonlinear, non-decreasing, bounded (i.e. $\mathcal{F}_i$ is expected to approach finite maximum values asymptotically) and

piece-wise differentiable functional form. For computational efficiency it is desirable that its derivative be easy to compute.

If inputs are continuous, assuming values on [0, 1], generally the logistic function chosen is:

$$\mathcal{F}_i(v_i) = \frac{1}{1+\exp(-\beta\, v_i)} \tag{5}$$

where β denotes the slope parameter which has to be a priori chosen. In the limit, as β approaches infinity, the logistic function becomes simply a threshold function.

5 Network Topologies

In the previous section, we discussed the characteristics of the basic processing element in a CNN. This section focuses on the pattern of connections between the processing elements (termed architecture) and the propagation of data. As for the pattern of connection a major distinction can be made between computational feedforward neural networks, and recurrent computational neural networks. Feedforward computational neural networks are networks that do not contain directed cycles. It is often convenient to organise the nodes of a feedforward CNN into layers. We define a l-layer feedforward network as a CNN where processing units are grouped into $l + 1$ subsets (layers) $L_0, L_1, ..., L_l$ such that if u in a layer L_a is connected to cell u_i in a layer L_b than $a < b$. For a strictly l-layer network, we require additionally that units can be connected only to units in the next layer, i.e. $b = a + 1$. All units in a layer L_0 are input units, all trainable units are in layers $L_1, ..., L_l$, and all units in layer L_l are output units.

In the simplest form of a layered feedforward CNN, we just have a layer L_0 of input nodes that projects on to a layer L_1 of output computational units. Such a CNN is called a single-layer feedforward network, with the designation 'single-layer' referring to the output layer of PEs. In other words, we do not count the layer of input nodes, because no computation is performed here (i.e. the transfer function being the identity). Many real world problems, however, need more sophisticated architectures to be adequate, even though interesting theoretical results can be obtained from these simple CNNs.

The second class of feedforward CNNs is distinguished by the presence of one or more hidden layers whose processing elements are correspondingly called hidden elements or units. The function of the hidden units is to intervene between the external input and the network output to extract higher order statistics.

The nodes of the input layer L_0 supply the respective elements of the input vector which constitute the input signals applied to the processing elements of the second layer (i.e. the first hidden layer L_1). The output signals of L_1 are used as the inputs to L_2, and so on for the rest of the network. Characteristically, the processsing elements in each layer are assumed to have the same transfer function φ and

to have the output signals of the preceding layer as their inputs. The set of output signals of the processing elements in the output layer L_1 constitutes the overall response of the CNN.

The architectural graphs of Figure 3 illustrate the layout of two multi-layer feedforward CNNs for the case of a single hidden layer, so-called single hidden layer feedforward networks. For brevity, the networks of Figure 3 are referred to as 8:4:2 networks in that they have eight input nodes, four hidden processing elements and two output units.

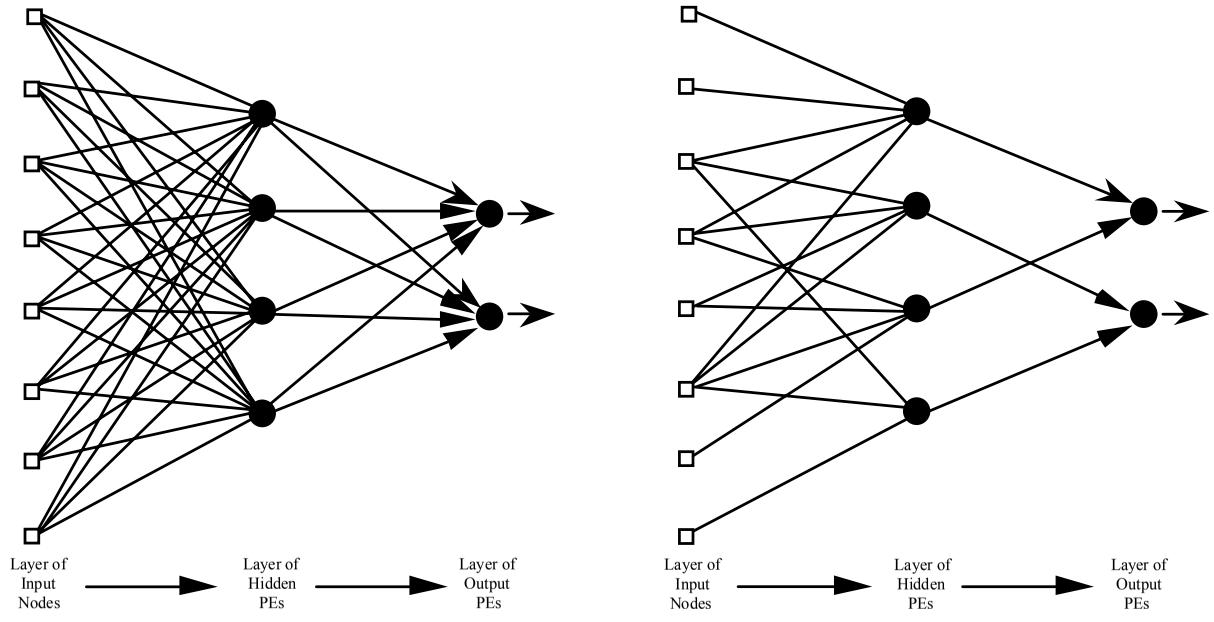
The CNN of Figure 3(a) is said to be fully connected in the sense that every node of the network is connected to every other node in the adjacent forward layer. If some of the connections are missing from the network, then the network is said to be partially connected. An example of such a CNN is shown in Figure 3(b). Each processing element in the hidden layer is connected to a partial set of input nodes in its immediate neighbourhood. Such a set of localised nodes feeding a processing element is said to constitute the receptive field of the processing element. The CNN of Figure 3(b) has the same number of input nodes, hidden units and output units as Figure 3(a), but it has a specialised structure. In real world applications, the specialised structure built into the design of a feedforward CNN reflects prior information about the problem to be analysed. The procedure for translating a feedforward CNN diagram, such as that shown in Figure 3, into the corresponding mathematical function Φ, follows from a straightforward extension of Section 3.

The output of unit i is obtained by transforming the net input, as in Equation (4), with a nonlinear activation function $\mathcal{F}_i$ to give

$$u_i = \mathcal{F}_i\left(\sum_{j=0,1,\ldots,k} w_{ij}\, u_j\right) \tag{6}$$

where the sum runs over all inputs and units which send connections to unit i (including a bias parameter). For a given set of values applied to the inputs of the CNN, successive use of expression (6) allows the activation of all processing elements in the CNN to be evaluated including those of the output units. This process can be regarded as a forward propagation of signals through the network. The result of the whole computation is well-defined and we do not have to deal with the task of synchronising the processing elements. We just assume that the processing elements are visited in a fixed order, each processing element re-evaluating and changing its activation before the next one is visited.

A *recurrent* (*feedback*) *computational neural network* distinguishes itself from a feedforward CNN in that it contains cycles (i.e. feedback connections) as illustrated by the architectural graph in Figure 4. The data is not only fed forward but also back from output to input units. This recurrent structure has a profound impact on the learning capability of the CNN and its performance. In contrast to feedforward networks, the computation is not uniquely defined by the interconnection pattern and the temporal dimension must be considered.



(a) Fully connected CNN

(b) Partially connected CNN

Figure 3 Feedforward computational neural network architectures with one hidden layer: (a) fully connected and (b) partially connected (see Fischer 1998)

When the output of a processing element is fed back to the same element, we are dealing with a recursive computation without an explicit halting condition. We must then define what to expect from the CNN. Is the fixed point of the recursive evaluation the desired result or one of the intermediate computations? To solve this problem it is usually assumed that every computation takes a certain amount of time at each node. If the arguments for a processing element have been transmitted at time t, its output will be produced at time $t + 1$. A recursive computation can be stopped after a certain number of steps and the last computed output takes as the result of the recursive computation.

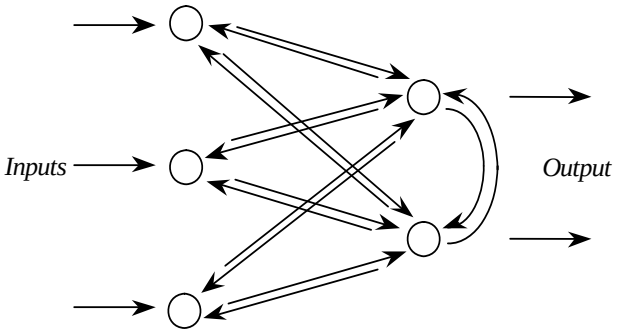


Figure 4 A recurrent [feedback] computational neural network architecture (see Fischer 1998b)

6 Learning in a Computational Neural Network

In addition to the characteristics of the processing elements and the network topology, the learning properties are an important distinguishing characteristic of different computational neural networks. In the context of CNNs, the process of learning may be best viewed as a search (typically local step-wise and steepest-gradient-based) in a multidimensional weight space for a solution (i.e. a set of weights) which gradually optimises a prespecified objective (performance, criterion, error) function with or without constraints. Learning is normally accomplished through an adaptive procedure, known as learning or training rule, or (machine) learning algorithm.

Each CNN is associated with one or more algorithms for machine learning. The input to these algorithms consists of a finite set of training examples, also called training patterns, $\{\boldsymbol{E}^r\}$, which is an n-vector of values that gives settings for the corresponding units of a CNN model, as illustrated in Figure 5. The jth component of an example, $\boldsymbol{E}_j^r$ is assigned to input unit u_i ($j = 1, ..., n$).

It is convenient to distinguish two types of learning situations: first, supervised learning problems, and second, unsupervised learning problems. For *supervised learning problems* (also known as learning with a teacher or associative learning)

each example $\boldsymbol{E}^r$ is associated with a correct response $\boldsymbol{C}^r$ (also termed teacher signal) for the CNN's output units. For CNNs with more than one output unit $\boldsymbol{C}^r$ is a vector with components $\boldsymbol{C}_i^r$ (= 1, ..., m). For supervised learning problems the term training example [pattern] is intended to include both the input $\boldsymbol{E}^r$ and the desired response $\boldsymbol{C}^r$.

For unsupervised learning problems there are no specified correct network responses available. Unsupervised learning (typically based on some variation of Hebbian and/or competitive learning) generally involves the clustering of, or detection of similarities among, unlabelled patterns of a given training set. The idea here is to optimise some performance (criterion) function defined in terms of the output activity of the processing elements in the CNN. The weights and the outputs of the CNN are usually expected to converge to representations that capture the statistical regularities of the training data. Usually the success of unsupervised learning hinges on some appropriately designed CNN that encompasses a task-independent criterion of the quality of representation the CNN is required to learn.

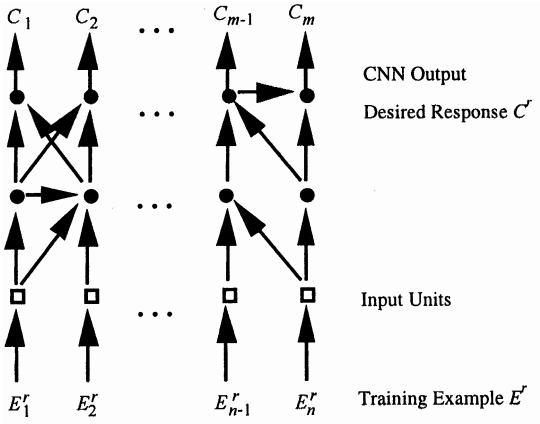


Figure 5 Training example inputs and outputs for a supervised learning model (see Fischer 1998b)

There is a wide variety of learning algorithms, for solving both supervised and unsupervised learning problems, that have been designed for specific network architectures. Many learning algorithms – especially those for supervised learning in feedforward CNNs – have their roots in function-minimisation algorithms that can be classified as local or global search (minimisation) algorithms. Learning algorithms are termed local if the computations needed to update each weight of a CNN can be performed using information available locally to that weight. This requirement may be motivated by the desire to implement learning algorithms in parallel hardware. Local minimisation algorithms, such as those based on gradient-descent, conjugate-gradient and quasi-Newton methods, are fast but usually converge to local minima. In contrast, global minimisation algorithms, such as simulated annealing and evolutionary algorithms, have heuristic strategies

to help escape from local minima. All these algorithms are weak in either their local or their global search. For example, gradient information useful in local search is not used well in simulated annealing and evolutionary algorithms. In contrast, gradient-descent algorithms with multistarts are weak in global search.

Designing efficient algorithms for CNN learning is a very active area of research. In formulating CNN solutions for (large-scale) real world problems, we seek to minimise the resulting algorithmic complexity and, therefore, the time required for a learning algorithm to estimate a solution from training patterns.

7 A Taxonomy of Computational Neural Networks

A taxonomy of four important families of computational neural network models (backpropagation networks, radial basis function networks, supervised and unsupervised ART models, and self-organising feature map networks) that seem particularly attractive for solving real world spatial analysis problems is presented in Figure 6. This taxonomy is first divided into CNNs with and without directed cycles. Below this, CNNs are divided into those trained with and without supervision. Further differences between CNNs not indicated in Figure 6 refer to the retrieval mode (synchronous versus asynchronous) and the inputs (discrete versus continuous).

Backpropagation computational neural networks have emerged as major workhorses in spatial analysis. They can be used as universal function approximators in areas such as spatial regression, spatial interaction, spatial choice and space-time series analysis, as well as pattern classifiers in data-rich environments (see, for example, Fischer and Gopal 1994a, Fischer et al. 1994, Leung 1997a). Strictly speaking, backpropagation is a technique which provides a computationally efficient procedure for evaluating the derivatives of the network's performance function with respect to the network parameters, and corresponds to a propagation of errors backwards through the network. This technique was popularised by Rumelhart et al. (1986).

Backpropagation is used primarily with multi-layer feedforward networks (also termed multi-layer perceptrons), so it is convenient to refer to this type of supervised feedforward network as a backpropagation network. Backpropagation CNNs are characterised by the following network, processing element and learning properties:

- *network properties*: multi-layer (typically single hidden layer) network with

- *processing element properties*: characteristically continuous inputs and continuous nonlinear sigmoid-type processing element transfer functions, normally assuming values [0, 1] or [–1, + 1], where the evaluation of the network proceeds according to the PE ordering, with each processing element computing and posting its new activation value before the next processing element is examined; the output unit activations are interpreted as the outputs for the entire CNN;

- *learning properties*: the essential ingredient is the backpropagation technique typically, but not necessarily, in combination with a gradient descent based learning algorithm.

The *Radial Basis Function CNNs* are a special class of a single hidden layer feed-forward network. Each processing element in the hidden layer utilises a *radial basis function*, such as a Gaussian kernel, as the transfer function. The argument of the transfer function of each hidden unit computes the Euclidean norm between the input vector and the centre of the unit. The kernel function is centred at the point specified by the weight vector associated with the processing element. Both the positions and the widths of these kernels must be learned from training patterns. Each output unit implements characteristically a *linear* combination of these radial basis functions. From the point of view of function approximation, the hidden units provide a set of functions that constitute a basis set for representing input patterns in the space spanned by the hidden units. There is a variety of learning algorithms for the Radial Basis Function CNNs. The basic one utilises hybrid learning that decouples learning at the hidden layer from learning at the output layer.

This technique estimates kernel positions and kernel widths using a simple unsupervised *k*-means based clustering algorithm, followed by a supervised least mean square algorithm to determine the connection weights between the hidden and the output layer. Because the output units are typically linear, a non-iterative algorithm can be used. After this initial solution is obtained, a supervised gradient-based algorithm can be used in a further step to refine the connection parameters. It is worth noting that Radial Basis Function Networks require more training data and more hidden units than backpropagation networks to achieve the same level of accuracy, but are able to train much faster – by several orders of magnitude.

ART network models differ from the previous CNNs in that they are recurrent. The data are not only fed forward but also back from the output to the input units. ART networks, moreover, are biologically motivated and were developed as possible models of cognitive phenomena in humans and animals. The basic principles of the underlying theory of these networks, known as adaptive resonance theory (ART), were introduced by Grossberg (1976a, b). They are essentially clustering CNNs with the task to automatically group unlabeled input vectors into several categories (clusters) so that each input is assigned a label corresponding to a unique cluster. The clustering process is driven by a similarity measure, vectors in the same cluster are similar which means that they are close to each other in the input space.

The networks use a simple representation in which each cluster is represented by the weight vector of a prototype unit. If an input vector is close to a prototype, then it is considered a member of the prototype's cluster, and any differences are attributed to unimportant features or to noise. The input and the stored prototype are said to resonate when they are sufficiently similar. There is no set number of

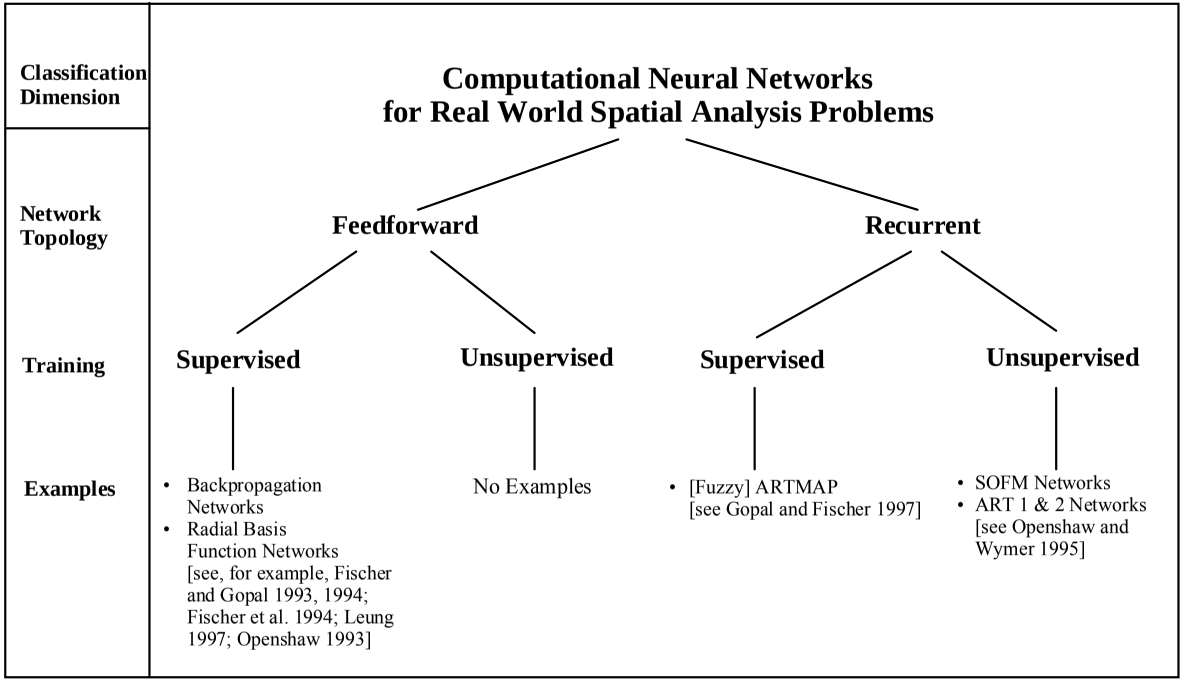


Figure 6 A taxonomy of computational neural networks

clusters, they are created as needed. Any clustering algorithm that does not prespecify the number of clusters must have some parameter that controls cluster granularity. For ART models this parameter is called the vigilance parameter. Regardless of its setting, the ART networks are stable for a finite set of training examples in that final clusters will not change with additional iterations from the original set of training examples. Thus, they show incremental clustering capabilities and can handle an infinite stream of input data. They do not require large memory to store training data because their cluster prototype units contain implicit representation of all the inputs previously encountered. However, ART networks are sensitive to the presentation order of the training examples.

They may yield different clustering on the same data when the presentation order of patterns is varied. Similar effects are also present in incremental versions of classical clustering techniques such as *k*-means clustering (which is also sensitive to the initial choice of cluster centers).

The basic architecture of an ART network involves three sets of processing elements:

- an input processing field (called the F_1-layer) consisting of two parts: the input portion with input nodes and the interface portion (interconnection),
- the layer of linear units (called the F_2-layer) representing prototype vectors whose outputs are acted as on competitive learning (i.e. the winner is the node with the weight vector which is closest, in a Euclidean distance sense, to the input vector), and
- supplementary units implementing a reset mechanism to control the degree of similarity of patterns placed on the same cluster.

The interface portion of the F_1-layer combines signals from the input portion and the F_2-layer in order to compare the similarity of the input signals to the weight vector for the cluster that has been selected as a candidate for learning. Each unit in the interface portion is connected in the F_2-layer by feedforward and feedback connections. Changes in the activation of units and in weights are governed by coupled differential equations.

ART models come in several varieties, most of which are unsupervised. The simplest are ART 1 designed for clustering training examples $\{\boldsymbol{E}^r\}$ with discrete data, $\boldsymbol{E}_j^r \in \{0, 1\}$ (Carpenter and Grossberg 1987a) and ART 2 designed for continuous data $\boldsymbol{E}_j^r \in [0, 1]$ (Carpenter and Grossberg 1987b). A more recent addition to the ART family is ARTMAP, a supervised learning version of ART, consisting of a pair of ART1 modules for binary training and teaching signals linked together via an inter-ART associative memory, called a map field (Carpenter et al. 1991). Fuzzy ARTMAP is a direct generalisation of ARTMAP for continuous data, achieved by replacing ART 1 in ARTMAP with fuzzy ART. Fuzzy ART synthesises fuzzy logic and adaptive resonance theory by exploiting the formal similarity between the computations of fuzzy subsethood and the

dynamics of prototype choice, search and learning. This approach is appealing because it nicely integrates clustering with supervised learning on the one hand and fuzzy logic and adaptive resonance theory on the other. Chapter 10 in this volume illustrates fuzzy ARTMAP performance in relation to backpropagation and maximum likelihood classifiers in a real world setting of a spectral pattern recognition problem.

Another important class of recurrent networks are self-organising feature map (SOFM) networks that have been foremost and primarily developed by Kohonen (1982, 1988). One motivation for such topology-preserving maps is the structure of the manmalian brain. The theoretical basis of these networks is rooted in vector quantisation theory, the motivation for which is dimensionality reduction. The purpose of SOFM is to map input vectors $\boldsymbol{E}^r \in \mathcal{R}^n$ onto an array of units [normally one- or two-dimensional] and to perform this transformation adaptively in a topologically ordered fashion such that any topological relationship among input vectors is preserved and represented by the network in terms of a spatial distribution of unit activities. The more these two vectors are related in the input space, the closer one can expect the position of the two units representing these input patterns to be in the array. The idea is to develop a topographic map of the input vectors so that similar input vectors would trigger nearby units. Thus, a global organisation of the units is expected to emerge. The essential characteristics of such networks may be summarised as follows:

- *Network properties*: a two-layer network where the input layer is fully connected to an output layer (also called a *Kohonen layer*) whose units are arranged in a two-dimensional grid (map) and are locally interacting (local interaction means that changes in the behaviour of a unit directly affect the behaviour of its immediate neighbourhood),

- *Processing element properties*: each output unit is characterised by a *n*-dimensional weight vector; processing elements are linear because each Kohonen unit computes its net input in a linear fashion with respect to its inputs; nonlinearities come in when the selection is made as to which unit 'fires'.

- *Learning properties*: unsupervised learning, that consists of adaptively modifying the connection weights of a network of locally interacting units in response to input excitations and in accordance with a competitive learning rule (i.e. weight adjustment of the winner and its neighbours); the weight adjustment of the neighbouring processing elements is instrumental in presserving the order of the input space.

SOFM networks can also be used as a front end for pattern classification or other decision-making processes where the Kohonen layer output can be fed into, for example a hidden layer of a backpropagation network.

It is hoped that the different families of CNN models presented in this chapter will give the reader an appreciation of the diversity and richness of these models. There is no doubt that they provide extremely rich and valuable nonlinear mathematical tools for spatial analysts. Their application to real world settings holds the potential for making fundamental advances in empirical understanding across a broad spectrum of application fields in spatial analysis.

8 Outlook – How Do Neurocomputing Techniques Differ?

In practice the vast majority of neural network applications are run on single-processor digital computers, although specialist parallel hardware is being developed. Neurocomputing techniques are frequently very slow and a speed-up would be extremely desirable, but parallelisation on real hardware has proved to be a non-trivial problem. A considerable speed-up could be achieved by designing better learning [training] algorithms using experience from other fields.

The traditional methods of statistics are either *parametric*, based on a family of models with a small number of parameters, or *non-parametric* in which case the models are totally flexible. One of the impacts of computational neural network models on spatial data analysis has been to emphasise the need in large-scale practical problems for something in between – families of models, but not the unlimited flexibility given by a large number of parameters. The two most widely used neural network architectures, multi-layer perceptrons and radial basis functions, provide two such families.

Another major difference is the emphasis on *on-line* procedures where data are not stored, except through the changes made by the learning algorithm. The theory of such algorithms has been studied for very long streams of data patterns, but the practical distinction is less clear, as such streams are constructed either by repeatedly cycling through the training set or by sampling the training patterns (with replacement). In contrast, procedures that utilise all data patterns together are called *batch* techniques. It is often forgotten that there are intermediate positions (called *epoch-based* techniques) using small batches chosen from the training set.

The most important challenges for the future are, firstly, to develop application domain-specific methodologies relevant for spatial analysis; secondly, to gain deeper theoretical insights into the complex relationship between training and generalisation, which is crucial for the success of real world applications; and, thirdly, to deliver high performance computing on neurohardware to enable rapid CNN prototyping, with the ultimate goal of developing application domain-specific CNN-systems which are automatic. This is crucial if CNNs are to become another regular element in the toolbox of spatial analysts.

References

Baumann J.H., Fischer M.M. and Schubert U. (1983): A multiregional labour supply model for Austria: The effects of different regionalisations in multiregional labour market modelling, *Papers of the Regional Science Association* 52, 53-83

Benediktsson J.A. and Kanellopoulos I. (1999): Classification of multisource and hyperspectral data based on decision fusion, *IEEE Transactions on Geoscience and Remote Sensing* 37 (3), 1367-1377

Benediktsson J.A., Swain P.H. and Ersoy O.K. (1990): Neural network approaches versus statistical methods in classification of multisource remote sensing data, *IEEE Transactions on Geoscience and Remote Sensing* 28, 540-552

Bezdek J.C. (1994): What's computational intelligence? In: Zurada J.M., Marks II R.J. and Robinson C.J. (eds.) *Computational Intelligence: Imitating Life*, IEEE Press, Piscataway [NJ], pp. 1-12

Carpenter G.A. and Grossberg S. (eds.) (1991): *Pattern Recognition by Self-Organising Neural Networks*, MIT Press, Cambridge [MA]

Carpenter G.A. and Grossberg S. (1987a): A massively parallel architecture for a self-organising neural pattern recognition machine, *Computer Vision, Graphics, and Image Processing* 37, 54-115

Carpenter G.A. and Grossberg S. (1987b): ART 2 stable self-organising of pattern recognition codes for analog input patterns, *Applied Optics* 26, 4919-4930

Carpenter G.A., Grossberg S. and Reynolds J.H. (1991) ARTMAP supervised real-time learning and classification of nonstationary data by a self-organising neural network, *Neural Networks* 4, 565-588

Fischer M.M. (2001): Spatial analysis. In: Smelser N.J. and Baltes P.B. (eds.) *International Encyclopedia of the Social & Behavioural Sciences*, Elsevier, Amsterdam, pp. 14752-14758

Fischer M.M. (2000): Methodological challenges in neural spatial interaction modelling: The issue of model selection. In: Reggiani A. (ed.) *Spatial Economic Science: New Frontiers in Theory and Methodology*, Springer, Berlin, Heidelberg, New York, pp. 89-101

Fischer M.M. (1999a): Intelligent GI analysis. In: Stillwell J.C.M., Geertman S. and Openshaw S. (eds.) *Geographical Information and Planning*, Springer, Berlin, Heidelberg, New York, pp. 349-368

Fischer M.M. (1999b): Spatial analysis: Retrospect and prospect. In: Longley P.A., Goodchild M.F, Maguire P. and Rhind D.W. (eds.) *Geographical Information Systems: Principles, Technical Issues, Management Issues and Applications*, John Wiley, Chichester [UK], New York, pp. 283-292

Fischer M.M. (1998a): Computational neural networks: An attractive class of mathematical models for transportation research. In: Himanen V., Nijkamp P. and Reggiani A. (eds.) *Neural Networks in Transport Applications*, Ashgate, Aldershot, pp. 3-20

Fischer M.M. (1998b): Computational neural networks. A new paradigm for spatial analysis, *Environment and Planning A* 30 (10), 1873-1892

Fischer M.M. (1995): Fundamentals in neurocomputing. In: Fischer M.M., Sikos T. and Bassa L. (eds.) *Recent Developments in Spatial Information, Modelling and Processing*, Geomarket, Budapest, pp. 31-41

Fischer M.M. and Abrahart R.J. (2000): Neurocomputing. Tools for geographers. In: Openshaw S., Abrahart R.J. and Harris T. (eds.) *GeoComputation*, Taylor & Francis, London, New York, pp. 187-217

Fischer M.M. and Getis A. (eds.) (1997): *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg, New York

Fischer M.M. and Gopal S. (1996): Spectral pattern recognition and fuzzy ARTMAP: Design features, system dynamics and real world simulations. In: *Proceedings of the Fourth European Congress on Intelligent Technologies and Soft Computing* [EUFIT'96], Elite Foundation, Aachen, pp. 1664-1668

Fischer M.M. and Gopal S. (1994a): Artificial neural networks. A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34, 503-527

Fischer M.M. and Gopal S. (1994b): Neurocomputing and spatial information processing. In: *Proceedings of the Eurostat/DOSES Workshop on* '*New Tools for Spatial Data Analysis*', Lisbon, Eurostat, Luxembourg, pp. 55-68

Fischer M.M. and Gopal S. (1993): Neurocomputing – A new paradigm for geographic information processing, *Environment and Planning* A 25 (6), 757-760

Fischer M.M. and Leung Y. (1998): A genetic-algorithm based evolutionary computational neural network for modelling spatial interaction data, *The Annals of Regional Science* 32 (3), 437-458

Fischer M.M. and Staufer P. (1999): Optimisation in an error backpropagation neural network environment with a performance test on a spectral pattern classification problem, *Geographical Analysis* 31 (2), 89-108

Fischer M.M., Hlaváckova-Schindler K. and Reismann M. (1999): A global search procedure for parameter estimation in neural spatial interaction modelling, *Papers in Regional Science* 78, 119-134

Fischer M.M., Gopal S., Staufer P. and Steinnocher K. (1994): Evaluation of neural pattern classifiers for a remote sensing application. Paper presented at the 34th European Congress of the Regional Science Association, Groningen, August 1994 [published 1997 in *Geographical Systems* 4 (2), 195-223]

Fogel D.B. (1995): *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway [NJ]

Gallant S.I. (1993): *Neural Network Learning and Expert Systems*, MIT Press, Cambridge [MA]

Goldberg D.E. (1989): *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, Reading [MA]

Gopal S. and Fischer M.M. (1997): Fuzzy ARTMAP – A neural classifier for multispectral image classification. In: Fischer M.M. and Getis A. (eds.) *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling and Computational Intelligence,* Springer, Berlin, Heidelberg, New York, pp. 306-335

Gopal S. and Fischer M.M. (1996): Learning in single hidden layer feedforward network models, *Geographical Analysis* 28 (1), 38-55

Gopal S. and Fischer M.M. (1993): Neural net based interregional telephone traffic models. In: *Proceedings of the International Joint Conference on Neural Networks* [IJCNN'93], Nagoya, Japan, pp. 2041-2044

Gopal S. and Woodcock C. (1996): Remote sensing of forest change using artificial neural networks, *IEEE Transactions on Geoscience and Remote Sensing* 34, 398-403

Grossberg S. (1976a): Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors, *Biological Cybernetics* 23, 121-134

Grossberg S. (1976b): Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction and illusion, *Biological Cybernetics* 23, 187-202

Haining R.P. (1994): Designing spatial data analysis modules for geographical information systems. In: Fotheringham S. and Rogerson, P. (eds.) *Spatial Analysis and GIS*, Taylor & Francis, London, pp. 45-63

Hlavácková-Schindler K. and Fischer M.M. (2000): An incremental algorithm for parallel training of the size and the weights in a feedforward neural network, *Neural Processing Letters* 11 (2), 131-138

Holland J.H. (1975): *Adaptation in Natural and Artificial Systems*, University of Michigan, Ann Arbor

Kim T.J., Wiggins L.L. and Wright J.R. (eds.) (1990): *Expert Systems: Applications to Urban and Regional Planning*, Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 191-201

Kohonen T. (1988): *Self-Organisation and Associative Memory*. Springer, Berlin, Heidelberg, New York [1rst edition 1984]

Kohonen T. (1982): Self-organised formation of topologically correct feature maps, *Biological Cybernetics* 43, 59-69

Leung Y. (1997a): *Intelligent Spatial Decision Support Systems*. Springer, Berlin, Heidelberg, New York

Leung Y. (1997b): Feedforward neural network models for spatial data pattern classification. In: Fischer M.M. and Getis A. (eds.) *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg, New York, pp. 336-359

Leung Y. (1993): Towards the development of an intelligent support system. In: Fischer M.M. and Nijkamp P. (eds.) *Geographic Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 131-145

Leung Y., Dong T.X. and Xu Z.B. (1998): The optimal encoding of biased association in linear associative memories, *Neural Networks* 11, 877-884

Leung Y., Zhang J.S. and Xu Z.B. (1997): Neural networks for convex hull computation, *IEEE Transactions on Neural Networks* 8, 601-611

Liu Y. and Yao X. (1999a): Ensemble learning via negative correlation, *Neural Networks* 12, 1391-1398

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

Moddy A., Gopal S. and Strahler A.H. (1996): Artificial neural network response to mixed pixels in coarse-resolution satellite data, *Remote Sensing of the Environment* 58, 329-343

Openshaw S. (1995): Developing automated and smart spatial pattern exploration tools for geographical systems applications, *The Statistician* 44 (1), 3-16

Openshaw S. (1993): Modelling spatial interaction using a neural net. In: Fischer M.M. and Nijkamp P. (eds.) *Geographic Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 147-164

Openshaw S. and Abrahart R.J. (eds.) (2000): *GeoComputation*, Taylor & Francis, London, New York

Openshaw S. and Taylor P. (1979): A million or so correlation coefficients: Three experiments on the modifiable areal unit problem. In: Bennett R.J, Thrift N.J. and Wrigley, N. (eds.) *Statistical Applications in the Spatial Sciences*, Pion, London, pp. 127-144

Openshaw S. and Wymer C. (1995): Classifying and regionalising census data. In: Openshaw, S. (ed.) *Census Users Handbook*, Geoinformation International, Cambridge, pp. 353-361

Openshaw S., Fischer M.M., Benwell G. and Macmillan B. (2000): GeoComputation research agendas and futures. In: Openshaw S. and Abrahart R.J. (eds.) *GeoComputation*, Taylor & Francis, London, pp. 379-400

Rumelhart D.E., Hinton G.E. and Williams R.J. (1986): Learning internal representations by error propagations. In: Rumelhart D.E., McClelland J.L. and the PDP Research Group (eds.) *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, MIT Press, Cambridge [MA], pp. 318-362

Smith T.R., Peuquet D., Menon S. and Agarwal P. (1987): KBGIS-II. A knowledge based geographical information system, *International Journal of Geographic Information Systems* 1, 149-172

Tibshirani R. (1996): A comparison of some error estimates for neural network models, *Neural Computation* 8 (1), 152-163

Turton I., Openshaw S. and Diplock G.J. (1997): A genetic programming approach to building new spatial model relevant to GIS. In: Kemp Z. (ed.) *Innovations in GIS 4*, Taylor & Francis, London, pp. 89-102

Weigend A.S., Huberman B. and Rumelhart D.E. (1990): Predicting the future: A connectionist approach, *International Journal of Neural Systems* 1, 193-209

White H. (1990): Connectionist nonparametric regression: Multi-layer feedforward networks can learn arbitrary mappings, *Neural Networks* 3, 535-550

Wilkinson G.G., Fierens F. and Kanellopoulos I. (1995): Integration of neural and statistical approaches in spatial data classification, *Geographical Systems* 2, 1-20

Wise S., Haining R. and Ma J. (1996): Regionalisation tools for the exploratory spatial analysis of health data. In: Fischer M.M. and Getis A. (eds.) *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg, New York, pp. 83-100

7 Artificial Neural Networks: A New Approach to Modelling Interregional Telecommunication Flows

with S. Gopal

This paper suggests a new modelling approach, based upon a general nested sigmoid neural network model. Its feasibility is illustrated in the context of modelling interregional telecommunication traffic in Austria and its performance is evaluated in comparison with the classical regression approach of the gravity type. The application of this neural network approach may be viewed as a three-stage process. The first stage refers to the identification of an appropriate network from the family of two-layered feedforward networks with three input nodes, one layer of (*sigmoidal*) *intermediate nodes and one* (*sigmoidal*) *output node. There is no general procedure to address this problem. We solved this issue experimentally. The input-output dimensions have been chosen in order to make the comparison with the gravity model as close as possible. The second stage involves the estimation of the network parameters of the selected neural network model. This is performed via the adaptive setting of the network parameters* (*training, estimation*) *by means of the application of a least mean squared error goal and the error back-propagating technique, a recursive learning procedure using a gradient search to minimise the error goal. Particular emphasis is laid on the sensitivity of the network performance to the choice of the initial network parameters as well as on the problem of overfitting. The final stage of applying the neural network approach refers to the testing of the interregional teletraffic flows predicted. Prediction quality is analysed by means of two performance measures, average relative variance and the coefficient of determination, as well as by the use of residual analysis. The analysis shows that the neural network model approach outperforms the classical regression approach to modelling telecommunication traffic in Austria.*

1 Introduction

Telecommunication like transportation interactions, take place over geographic space. But in contrast to the vast literature devoted to the analysis of region-to-region travel demand (see Fischer 1993), relatively little research effort has been directed towards region-to-region telecommunication demand. One reason for this deficiency in research is that telecommunication data are either not available (as in the case of most European countries) or viewed as proprietary by the private companies (as in the case of US) which provide the telecommunication services

(Guldmann 1992). A better understanding of the spatial structure of telecommunication interactions is becoming more and more important, especially in the context of Europe where the telecommunication sector is increasingly coming under debate due to deregulation trends. The knowledge of telecommunication linkages might also assist to clarifying issues such as the complementarity and substitution effects between transportation and telecommunication (Salomon 1986) and the value of telecommunication technologies as they influence patterns of regional development (see Gillespie and Williams 1988).

The general purpose of this paper is to set out an (artificial) neural network (connectionist) approach to modelling interregional telecommunication interactions and, thus, to contribute to the debate on the feasibility of neural network computing (neurocomputing) in geography and regional science (Openshaw 1992, Fischer and Gopal 1993). The specific objectives may be stated as follows: first, to develop and estimate a neural net telecommunication flow model with explicit treatment of geographical distance, second, to evaluate its prediction quality in relation to the classical regression model of the gravity type.

The general model suggested is based upon the family of two-layered feedforward neural networks with sigmoidal nonlinearities, the most tractable and most prominent family of artificial neural net models (see Hecht-Nielsen 1990). The value of this approach to modelling interregional telecommunications will be illustrated on the basis of an application using a new telecommunication data set of Austria. The application of the approach may be viewed as a three-stage process. The first stage refers to the identification of an appropriate model candidate from the family of two-layered feedforward networks. The input-output dimensions are chosen so as to make the comparison to the gravity model as close as possible. The second stage involves the estimation of the network parameters of the selected neural network model. This is performed via the adaptive setting of the network parameters by means of the application of a least mean squared error goal and the error backpropagating technique, a recursive learning procedure using a gradient search to minimise the error goal. Particular emphasis is placed on the sensitivity of the network performance to the choice of the initial network parameters as well as the problem of overfitting. The final stage of applying the approach refers to the testing of the interregional teletraffic flows predicted. Prediction quality is analysed by means of performance measures, as well as by the use of residual analysis, and compared with gravity model predictions.

The remainder of this paper is organised as follows. Section 2 briefly characterises the classical regression model of the gravity type that is used as a benchmark model in this study, then proceeds to outline the neural networks approach by considering it as a three-stage process, and lastly, describes the estimation process based upon a supervised learning algorithm. The experimental environment and the results of model identification, estimation and testing are discussed in relation to the classical telecommunications flow model in Section 3. Conclusions and areas for further research are outlined in Section 4.

2 The Methodological Framework

The literature on telecommunication flow modelling is primarily based on the conventional regression approach of the gravity type (Pacey 1983, Rietveld and Janssen 1990, Rossera 1990, Fischer et al. 1993, Guldmann 1992), and thus lies in the tradition of the broader and well-established framework of spatial interaction modelling in geography and regional science (see for example, Fotheringham and O'Kelly 1989). This conventional model approach serves as a benchmark for evaluating the performance of the alternative, the neural net approach. We first briefly characterise the conventional modelling approach, and then the general neural network model.

2.1 The Gravity Model as Benchmark

Let T_{rs} denote the intensity of telecommunication from region r to region s ($r, s = 1, ..., n$), measured in terms of erlang or minutes. Then T_{rs} is called interregional traffic, if $r \neq s$, and intraregional traffic, if $r = s$. The conventional approach applied to the problem of modelling telecommunications traffic belongs to the class of (unconstrained) gravity models. It is usually assumed (see Rietveld and Janssen 1990, Fischer et al. 1993) that

$$T_{rs}^{conv} = G(A_r, B_s, F_{rs}) \qquad r, s = 1, ..., n \tag{1}$$

with

A_r a factor associated with the region r of origin and representing the intensity of telephone communication generated by this region,

B_s a factor associated with the region s of destination and representing destination-specific pull factors or the degree to which the in-situ attributes of a particular destination attract telephone communication traffic, and

F_{rs} a factor associated with the origin-destination pairs (r, s) representing the inhibiting effect of geographic separation from region r to region s.

A_r and B_s are called mass or activity variables, F_{rs} is termed separation variable, provided it is specified in a way such that higher values imply less telecommunications traffic. In this approach, the specific form of the function to be fitted to the data is first chosen and then the fitting according to some error criterion (usually mean squared error) is carried out.

G is usually specified in such a way that the interregional telecommunication flow model reads as follows:

$$T_{rs}^{conv} = K\, A_r^{\alpha_1}\, B_s^{\alpha_2}\, F_{rs}(D_{rs}) \qquad r, s = 1, ..., n \tag{2}$$

with

$$F_{rs}(D_{rs}) = D_{rs}^{\alpha_3} \tag{3}$$

where T_{rs}^{conv} denotes the intensity of telecommunication from r to s predicted by (2) and (3). A_r and B_s represent the potential pool of calls in region r and the potential draw of calls in region s, respectively. We have decided to use gross regional product as a measure for A_r and B_s. Gross regional product as a proxy of economic activity and income is relevant for both business and private phone calls (see Rietveld and Janssen 1990). D_{rs} denotes distance from region r to region s. K is a scale parameter (constant), α_1, α_2 and α_3 are parameters to be estimated. n denotes the number of telecommunication regions.

The usual strategy to estimating Equations (2) and (3) is to assume that a normally distributed multiplicative error term ε_{rs} applies, i.e., $\varepsilon_{rs} \sim N(0,\sigma)$ independently of A_r, B_s and D_{rs}. In this case, ordinary least squares (OLS) regression can be applied after a logarithmic transformation yielding

$$\ln T_{rs}^{conv} = \ln K + \alpha_1 \ln A_r + \alpha_2 \ln B_s + \alpha_3 \ln D_{rs} + \tilde{\varepsilon}_{rs} \qquad r, s = 1, ..., n. \tag{4}$$

OLS estimation leads to unbiased and consistent estimates $\hat{\alpha}_1$, $\hat{\alpha}_2$ and $\hat{\alpha}_3$. This model version is usually called log-normal gravity model and will be used as benchmark to evaluate the relative efficiency of the neural net model to be developed below. It is worthwhile to note that there are some problems with this model. For example, it is based on the questionable assumption that the variances of the error terms are identical, and it cannot be used when some of the interaction flows are zero.

The standard method for assessing the goodness of fit of this regression model is through the use of R^2, the coefficient of determination. It is worth noting that the OLS estimator of the log-normal gravity model minimises the sum of squared residuals and, thus, automatically maximises R^2. Consequently, maximisation of R^2, as a criterion for the OLS estimator, is formally identical to the least squares criterion. An unsatisfactory fit in terms of R^2 may result from the failure to include all relevant explanatory variables, such as for example, barrier effects impeding the continuous flow pattern in space (see Fischer et al. 1993).

2.2 The General Neural Network Model

The general neural network approach set out in this section is based upon the family of two-layered feedforward neural networks with sigmoidal processing units. The two-layered feedforward neural network is a particular neural network

architecture characterised by a hierarchical design consisting of I_1 input nodes, one layer of I_2 intermediate (hidden) nodes and I_3 output nodes as displayed in Figure 1.

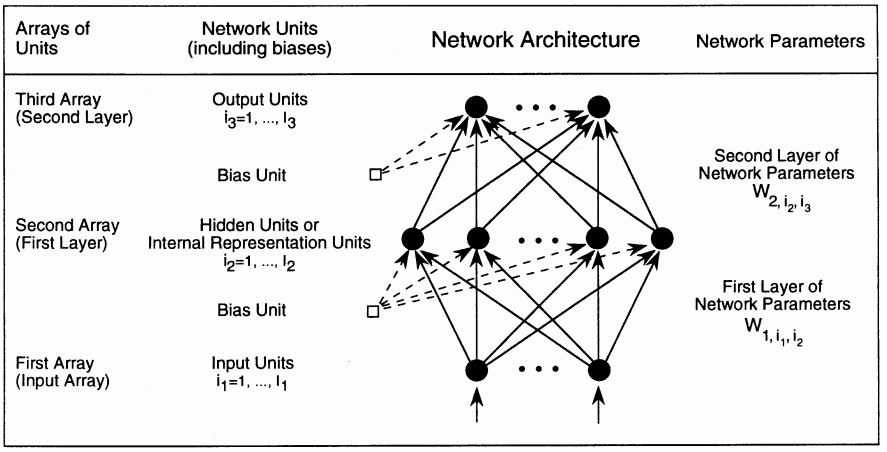


Figure 1 The general two-layer feedforward neural network model

The input units, $i_1 = 1, \ldots, I_1$, send (continuous-valued) signals Y_{1, i_1} to intermediate (hidden) units, indexed by $i_2 = 1, \ldots, I_2$, that process them in some characteristic way (linear summation of input signals multiplied by the corresponding network weights) and produce a bounded output signal Y_{2, i_2} that is sent via the second layer of connections to the output units, indexed by $i_3 = 1, \ldots, I_3$, which now treat the hidden units as inputs and produce the network output Y_{3, i_3}. Note also the presence of bias units. These output a fixed unit value and can be viewed of as constant terms in the equations defining the process carried out by the network. The network has two layers of unidirectional connections with which weights (parameters) are associated. The first layer of weights connects the input units to the hidden units, and the second layer connects the hidden units to the output units. The architecture is feedforward because signals flow in only one direction. The fundamental characteristic of this network architecture is that there are no connections leading from an unit to units in previous arrays nor to the other units in the same array, nor to the units more than one array ahead. The arrays of nodes are connected in such a way that the units are connected to every unit in the next array.

The general two-layer neural network model, denoted by $\mathcal{N}_{I_1, I_2, I_3}$ or $(I_1 : I_2 : I_3)$, may be described as follows:

$$X_{2,i_2} = \sum_{i_1=1}^{I_1} W_{1,i_1,i_2} Y_{1,i_1} + W_{1,I_1+1,i_2} \qquad i_2 = 1, ..., I_2 \tag{5}$$

$$Y_{2,i_2} = f(X_{2,i_2}) \qquad i_2 = 1, ..., I_2 \tag{6}$$

$$X_{3,i_3} = \sum_{i_2=1}^{I_2} W_{2,i_2,i_3}\, Y_{2,i_2} + W_{2,I_2+1,i_3} \qquad i_3 = 1, ..., I_3 \tag{7}$$

$$Y_{3,i_3} = f(X_{3,i_3}) \qquad i_3 = 1, ..., I_3 \tag{8}$$

with

I_j number of processing units of the jth array (j = 1, 2, 3),
i_j indices associated with the jth array of units (j = 1, 2, 3),
Y_{j,i_j} (continuous-valued) output of the units i_j of the jth array (j = 1, 2, 3),
X_{j,i_j} (continuous-valued) input to the processing unit i_j belonging to the jth array (j = 2, 3), [note: $X_{1,i_1} = Y_{1,i_1}$],
$W_{k,i_k,i_{k+1}}$ weights (connection weights, parameters) of the kth array between the kth and (k + 1)th array of units with k = 1; connection weights from the hidden to the output layer, k = 2; connection weights from the input array to the hidden layer [note: $W_{k,i_k,i_{k+1}}$ is a positive number if unit i_k excites unit i_{k+1} and a negative number if unit i_k inhibits unit i_{k+1}],
W_{1,I_1+1,i_2} bias unit of hidden units,
W_{2,I_2+1,i_3} bias unit of the output units.

where the (transfer) function f is the following logistic function of hidden and output units, i_2 and i_3:

$$f(X_{k,i_k}) = \frac{1}{1+\exp(-a\,X_{k,i_k})} \qquad k = 2, 3. \tag{9}$$

Equation (9) scales the activation of a unit sigmoidally between 0 and 1. Strict monotonocity implies that the activation derivative of f is positive: $df / dX_{k,i_k} = a\, f(1 - X_{k,i_k}) > 0$. This property increases the network's computational richness and facilitates noise suppression. The slope of the sigmoid, a, can be absorbed into weights and biases without loss of generality and is set to one.

The parameters of Equations (5) through (8) $\{W_{k,i_k,i_{k+1}}, i_k = 1, ..., I_k; k = 1, 2\}$ and the adaptable biases $\{W_{1,I_1+1,i_2}, W_{2,I_2+1,i_3}\}$ [briefly $\boldsymbol{W} \equiv \{W_{k,I_k+1,i_{k+1}}; k = 1, 2\}$] have to be estimated. The weights correspond to a point $\boldsymbol{w}$ in the $K = [\Sigma_{k=1}^{2}\, I_{k+1}(I_k + 1)]$-dimensional Euclidean space $\mathfrak{R}^K$. For every point $\boldsymbol{w}$ in the network configuration space $\Omega \subset \mathfrak{R}^K$, the network model (5) through (9) is a realisation of a deterministic mapping from an input, say $\boldsymbol{x} \in \boldsymbol{X} \subset \mathfrak{R}^{I_1}$, to an output, say $\boldsymbol{y} \in \boldsymbol{Y} \subset \mathfrak{R}^{I_3}$ (see Levin et al. 1990). We denote this mapping by $\boldsymbol{y} = \boldsymbol{F}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{F}_{\boldsymbol{w}} : \mathfrak{R}^{I_1} \times \Omega \to \mathfrak{R}^{I_3}$. Inserting (5) through (7) and (9) into (8) describes the general neural net model in the following compact form, for i_3 = 1, ..., I_3.

$$\boldsymbol{Y} \equiv Y_{3,i_3} = \left\{1+\exp\left[-\left(\sum_{i_2=1}^{I_2} W_{2,i_2,i_3} \left\{1+\exp\left[-\left(\sum_{i=1}^{I_1} W_{1,i_1,i_2} Y_{1,i_1} + W_{1,I_1+1,i_2}\right)\right]\right\}^{-1} + W_{2,I_2+1,i_3}\right)\right]\right\}^{-1} \equiv F_{\boldsymbol{w}}(\boldsymbol{X}) \qquad (10)$$

The general model defined by (10) can be viewed as an input-output model $\boldsymbol{Y} = F_{\boldsymbol{w}}(\boldsymbol{X})$ or as a nonlinear regression function with quite a specific form. Its usefulness as a flexible functional form hinges on the nature of the set of mappings from input to output spaces that it can approximate (White 1989). Recently, Hornik et al. (1989) and Cybenko (1989) have shown that networks of this type, with one layer of hidden units, can approximate any continuous input-output relation of interest to any desired degree of accuracy, provided sufficiently many hidden units are available (see also Hecht-Nielsen 1990). This result shows the appeal of (10).

2.3 Neural Network Modelling as a Three-Stage Process

The application of the general model (10) may be considered as a three-stage modelling process (see Figure 2). The first stage refers to the identification of a specific model from $\mathcal{N}_{I_1,I_2,I_3}$ for the problem at hand and involves the determination of I_1 , I_2 and I_3 . In order to make the comparison as close as possible with the gravity model, we identify the following subclass $\mathcal{N}_{I_1,I_2,I_3}$ defined as follows (see Figure 3):

(i) $I_1 = 3$ input units corresponding to the three independent variables A_r, B_s, and D_{rs} in (4),

(ii) $I_3 = 1$ output unit producing the neural network (telecommunication flow) prediction $Y_{3,1}$ $(= T_{rs}^{neur})$ to be compared with T_{rs}^{conv}, and

(iii) the number I_2 (hereafter abbreviated as I) of (sigmoidal) hidden units is a priori unknown.

There is no general procedure to determine I exactly. Consequently, identification and estimation overlap in the model building process. Here we employ the estimation procedure to carry out part of the identification and determine I experimentally by means of performance measures which indicate that specific model worthy of further investigation.

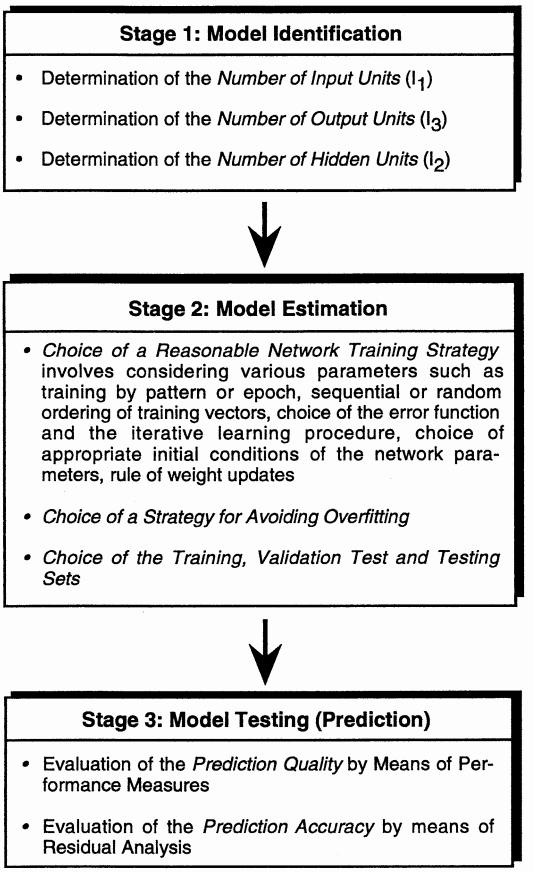


Figure 2 Neural network modelling as a three-stage process

The second stage refers to the estimation of the network parameters. Once an appropriate network model has been chosen, much of the effort in neural network building concerns the design of a reasonable network training (estimation) strategy. This necessitates engineering judgement in considering various parameters: training by pattern or epoch, sequential or random ordering of training vectors, choice of the error goal, and the iterative learning procedure, choice of appropriate initial conditions on the network parameters. The nonlinear character of (10) necessitates a computationally intensive iterative solution of determining the network parameters. As shown in what follows the estimation of the network parameters is performed via adaptive setting by means of the application of a least mean squared error goal and the error backpropagating technique, a recursive learning procedure using gradient search to minimise the error goal. A serious problem in model estimation is the problem of overfitting which is especially serious for noisy real world data of limited file length (see Section 3 for more details). The final stage involves the prediction of the interregional telecommunication flows

and evaluating the model's testing performance by means of performance measures and by the use of residual analysis, in relation to the benchmark model.

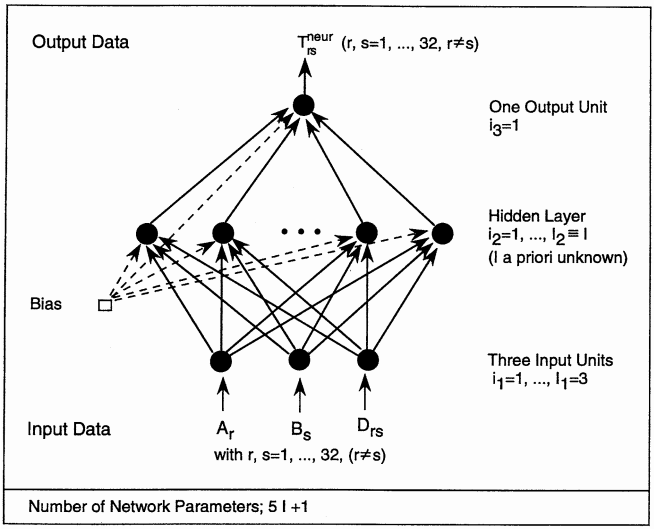


Figure 3 A subclass of the general two-layer feedforward neural network model to modelling telecommunications over space

2.4 The Training or Estimation Process

The problem of finding a suitable set $\boldsymbol{W}$ of parameters that approximate an unknown input-output relation $\boldsymbol{Y} = F(\boldsymbol{X})$ is solved using a supervised learning algorithm. Supervised learning requires a training set, that is, a set of input-output target (desired) examples, related through F, say $\xi^{(m)} \equiv \{\xi_l, 1 \le l \le m\}$, where $\xi \equiv (\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{x} \in \boldsymbol{X} \subset \Re^3$ and $\boldsymbol{y} \in \boldsymbol{Y} \subset \Re^1$ in our spatial interaction context. The relation F can be generally described by the probability density function defined over the space of input-output pairs $\boldsymbol{X} \otimes \boldsymbol{Y} \subset \Re^4$: $P_F(\xi) = P_F(\boldsymbol{x})\ P_F(\boldsymbol{y}|\boldsymbol{x})$, where $P_F(\boldsymbol{x})$ defines the region of interest in the input space and $P_F(\boldsymbol{y}|\boldsymbol{x})$ the functional (statistical) relation between the inputs and the outputs. The training set consists of examples drawn independently according to this probability density function.

Learning the training set by a model of the general model $\mathcal{N}_{3,\ I,\ 1}$ may be posed as a (nonlinear) optimisation problem by introducing a quality measure of the approximation of the desired relation F by the mapping F_w realised by the network (White 1989, Gyer 1992). In this study the additive error function for a set S of examples

$$E(S) \equiv E\left[\zeta(S)|\boldsymbol{w}\right] = \sum_{l \in S} e(\boldsymbol{y}_l \,|\, \boldsymbol{x}_l, \boldsymbol{w}) = \tfrac{1}{2} \sum_{l \in S} (\boldsymbol{y}_l - \hat{\boldsymbol{y}}_l)^2 \tag{11}$$

has been chosen which measures the dissimilarity between F and F_w on the restricted domain covered by a set S of input-output target data ($l < m$). The error function e ($\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}$) is a distance measure on $\Re^1$ between the target output (signal) $\boldsymbol{y}$ and the actual output $\hat{\boldsymbol{y}}$ of the network on the given input $\boldsymbol{x}$. The minimisation of E over the network's configuration space is called the training process. The task of learning, however, is to minimise that error for all possible examples related through F, namely, to generalise (Levin et al. 1990).

To accomplish this task, we utilise the error backpropagating technique of Rumelhart et al. (1986), a recursive learning procedure using a gradient search to minimise (11). Learning is carried out by iteratively adjusting the network parameters. This learning process is repeated until the network responds for each input vector $\boldsymbol{x}_l$ with an output vector $\hat{\boldsymbol{y}}_l$ that is sufficiently close to the target one $\boldsymbol{y}_l$.

Error backpropagating has three major components. In the first component, an input vector $\boldsymbol{x}_l = (x_{1l}, x_{2l}, x_{3l})$ is presented which leads via the forward pass to the activation of the network as a whole. This generates a difference (error) between the output of the network and the desired output. The second component computes the error factor (signal) for the output unit and propagates this factor successively back through the network (error backward pass). The third component computes the changes for the connection weights and the biases, and updates the parameters. This process continues until the network reaches a satisfactory level of performance.

Error backpropagation training may be performed on-line (i.e. after each presentation of an input signal; on-line learning) or off-line (i.e. after a set S of input presentations, off-line learning). We used the off-line learning mode, more precisely epoch training with epoch size of 20 input signals presented in random order (stochastic approximation) and parameter updates following the momentum update rule defined as (Rumelhart et al. 1986):

$$\Delta W_{k,i_k,i_{k+1}}(t+1) = -\,\eta\, \frac{\partial E(S)}{\partial W_{k,i_k,i_{k+1}}} + \gamma\, \Delta W_{k,i_k,i_{k+1}}(t) \tag{12}$$

where $\partial E(S) / \partial W_{k,i_k,i_{k+1}}$ denotes the partial derivative of $E(S)$ with respect to $W_{k,i_k,i_{k+1}}$ (the error gradient), η is the (constant) learning rate, γ ($0 \leq \gamma < 1$, the so-called momentum factor) is the relative contribution of the previous change of the parameter, and t represents the number of epochs, i.e., the number of times the network has been through a set S of 20 randomly chosen cases, after which the parameters are updated.

It is not easy to choose appropriate values for the learning rate parameter η and the momentum parameter γ for a problem at hand. A learning rate that is too large may cause the model to oscillate, and thereby to slow or prevent the network's convergence. The momentum term is much like the learning rate in that it is

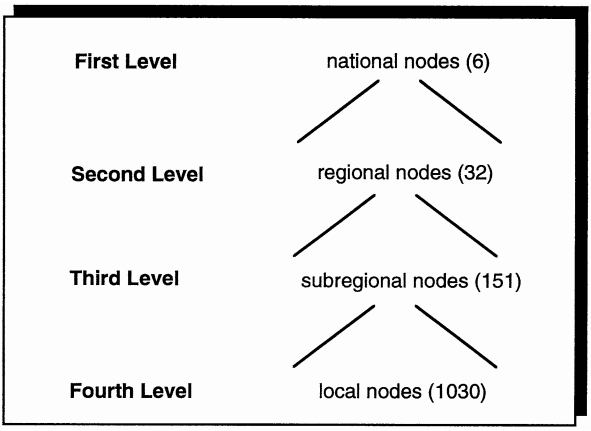
peculiar to specific error surface contours. The momentum term, if it overwhelms the learning rate, can make the system less sensitive to local changes. No rule for selecting optimal values for η and γ exists, although specific values are sometimes suggested in the literature. Experiments showed that it was best to use a small learning rate and a larger momentum term in our application setting. In all training cases, the learning rate parameter, η, was set to 0.15 and momentum, γ, to 0.8.

The backpropagating technique amounts to performing gradient descent on a hyper-surface in weight space Ω where at any point in that space the error of performance (11) is the height of the surface. The procedure, however, is not guaranteed to find a global minimum of E since gradient descent may get stuck in (poor) local minima. Thus, it makes sense to take different initial parameter values, and then select the estimate giving the smallest E. Although a number of more complex adaptation algorithms have been proposed to speed convergence (see Widrow and Lehr 1990, Shah et al. 1992), it seems unlikely that the complex regions formed by two-layer feedforward networks can be generated in few epochs when regions are disconnected. As all gradient descent procedures, back-propagation is sensitive to starting points (in the present context, the initial set of coupling strengths and biases in the network). The initial network parameters were drawn at random from an uniform distribution between –0.1 and 0.1. This random initialisation prevents the hidden units from acquiring identical weights during training. Five different random parameter initialisations (trials) were generated to analyse the robustness of the parameters and the network performance.

3 Experimental Environment and Results

3.1 Data, Training and Testing Results

The telecommunication data used in this study stem from network measurements of carried traffic in Austria in 1991, in terms of erlang, an internationally widely used measure of telecommunication contact intensity, which is defined as the number of phone calls (including facsimile transfers) multiplied by the average length of the call (transfer) divided by the duration of measurement. The data refer to the total telecommunication traffic between the 32 telecommunication districts representing the second level of the hierarchical structure of the Austrian telecommunication network (see Figure 4 and Figure 5). Due to measurement problems, intraregional traffic (i.e. $r = s$) is left out of consideration in this study.



Note: In several cases there are direct lines between regional nodes belonging to different national nodes. The same is true for subregional nodes belonging to different regional nodes.

Figure 4 The hierarchical structure of the Austrian telecommunication network

Thus, the data set for the model building process is made up of a set of $n(n-1) = 992$ 4-tuples $(A_r, B_s, D_{rs}; T_{rs}^{obs})$ with $r, s = 1, ..., 32$ and $r \neq s$. The first three components represent the input $\boldsymbol{x} = (x_1, ..., x_m, x_l = (x_{l1}, x_{l2}, x_{l3}), l = 1, ..., m)$, where I labels the individual training pairs, and the last component the desired (target) output (teacher) $\boldsymbol{y} = (y_l)$, displayed in Table 1.

Figure 5 The regional system ($n = 32$) for modelling interregional telecommunication in Austria

The input and output signals were preprocessed to logarithmically transformed data scaled into (0, 1), partly in order to make the comparison with the gravity model (4) as close as possible, and partly because experiments showed that it is easier for the model candidates to learn the logarithmically transformed data than the original raw data. The continuous-valued input signals were presented one point at a time to activate the network. The response generated at the output in response to each input was then used to calculate the error with respect to the teacher. The weights and biases were updated after each 20 patterns presented in random order (accumulated parameter correction). In one epoch the network sees each point from the training set exactly once. If the samples are chosen in random order, it also makes the path through weight-space stochastically, allowing wider exploration of the error surface.

Table 1 Input-target output pairs for training and testing the neural network model candidates

	Input components[a]			*Output*[b]
	A_r $(= x_{I1})$	B_s $(= x_{I2})$	D_{rs} $(= x_{I3})$	T_{rs}^{obs} $(= y_I)$
$\boldsymbol{x}_1$	A_1,	B_2,	$D_{1,2}$	$y_1 = T_{1,2}^{obs}$
$\boldsymbol{x}_2$	A_1,	B_3,	$D_{1,3}$	$y_2 = T_{1,3}^{obs}$
$\boldsymbol{x}_3$	A_1,	B_4,	$D_{1,4}$	$y_3 = T_{1,4}^{obs}$
⋮				⋮
$\boldsymbol{x}_{31}$	A_1,	B_{32},	$D_{1,32}$	$y_{31} = T_{1,32}^{obs}$
$\boldsymbol{x}_{32}$	A_2,	B_1,	$D_{2,1}$	$y_{32} = T_{2,1}^{obs}$
$\boldsymbol{x}_{33}$	A_2,	B_3,	$D_{2,3}$	$y_{33} = T_{2,3}^{obs}$
⋮				⋮
$\boldsymbol{x}_{961}$	A_{32},	B_1,	$D_{32,1}$	$y_{961} = T_{32,1}^{obs}$
⋮				⋮
$\boldsymbol{x}_{992}$	A_{32},	B_{31},	$D_{32,31}$	$y_{992} = T_{32,31}^{obs}$

[a] A_r represents the potential pool of calls in region r, measured in terms of gross regional product in r ($r = 1, ..., 32$); B_S represents the potential draw of calls in region s, measured in terms of the gross regional product in s ($s = 1, ..., 32$);

[b] T_{rs}^{obs} denotes the observed telecommunication flow from region r to region s ($r \neq s$), measured in terms of erlang.

3.2 Model Identification

After fixing I_1 and I_3 to 3 and 1 respectively, the identification process involves the determination of I_2 only, the issue of how many hidden units are needed. The choice of the number of hidden units in a feedforward structure design is not an easy task. It is generally problem dependent and often involves considerable engineering judgement. Often trade-offs between training time and modelling performance lead to iterative judgement of the network using simulation. For a given problem, the design of an adequately sized hidden layer is often not very obvious. Intuition suggests that 'the more the better' rule might be used to guide sizing the hidden layer, since the number of hidden units controls the model's flexibility. But very large sized hidden layers may become counterproductive. Too many free network parameters will allow the network to fit the training data arbitrarily closely, but will not necessarily lead to optimal predictions. There is no general procedure to determine the optimal size of the hidden layer for a particular task. A rule-of-thumb often cited is that the number of weights should be less than one tenth of the number of training patterns (Weigend et al. 1991). With a relatively small training set, this constraint is too restrictive.

The training set used in the identification stage of the model building process consists one-half of the available data, i.e. 496 patterns. However, this is not a hard rule. To determine I experimentally, i.e. the number of hidden units required, we considered 10 model candidates: (3:5:1), (3:10:1), (3:15:1), (3:20:1), (3: 25:1), (3:30:1), (3:35:1), (3:40:1), (3:45:1), and (3:50:1). Each model candidate was trained on the training set for 3000 epochs. The initial parameters were drawn at random from an uniform distribution between –0.1 and 0.1. Five different random initialisations were used to analyse variations due to different random initial conditions. The weights were updated after each 20 input signals as defined by Equation (12), presented in random order (stochastic approximation). The model identification process is based on the networks' overall performance, specified in terms of two measures. Performance of each model candidate was taken to be the average over a set of five performance values obtained from the five trials.

The first performance measure used is the average relative variance $ARV(S)$ of a set S of patterns, which is widely used in the neural network literature (see Weigend et al. 1991) defined as

$$ARV(S) = \frac{\sum_{l \in S} (\boldsymbol{y}_l - \hat{\boldsymbol{y}}_l)^2}{\sum_{l \in S} (\boldsymbol{y}_l - \bar{\boldsymbol{y}})^2} = \frac{1}{\hat{\sigma}^2} \frac{1}{N_S} \sum_{l \in S} (\boldsymbol{y}_l - \hat{\boldsymbol{y}}_l)^2 \tag{13}$$

where $\boldsymbol{y}_l$ denotes the target value and $\hat{\boldsymbol{y}}_l$ the actual network value, $\bar{\boldsymbol{y}}$ the average over the 20 desired values in S. The averaging, i.e. division by N_S [the number of patterns in set S (= epoch), N_S = 20] makes $ARV(S)$ independent of the size of the set. The division by the estimated variance $\hat{\sigma}^2$ of the data removes the dependence on the dynamic range of the data. This implies that if the estimated mean of the observed data would be taken as predictor, $ARV(S)$ would be equal to

unity (Weigend et al. 1991). In this study the variances of the individual sets *S* associated with the different epochs differ only slightly. Thus it appears to be reasonable to always use the variance of the entire data record $\hat{\sigma}^2 = \sigma_{all}^2 = 3.112$ as a proxy. A value of $ARV(S) = 0.1$ corresponds, thus, to an average absolute quadratic error of $ARV(S)(\sigma_{all}^2) = 0,1(3.112) = 0,3112 \approx 0,9682^2$. The alternative would have been to normalise each individual set *S* by its own variance.

Table 2 Model identification: Performance of selected model candidates from $\mathcal{N}_{3,I,1}$

Neural net model	*Number of parameters*	*Trial*[a]	*Performance measures*[b] *ARV*	R^2
3:5:1	26	1	0.0855	0.6163
		2	0.0898	0.6313
		3	0.0825	0.7585
		4	0.0803	0.7636
		5	0.0793	0.7658
		Av. Performance	0.0835 (0.0043)	0.7071 (0.0763)
3:10:1	51	1	0.0813	0.7613
		2	0.0793	0.7710
		3	0.0810	0.7698
		4	0.0788	0.7725
		5	0.0804	0.7685
		Av. Performance	0.0802 (0.0011)	0.7686 (0.0041)
3:15:1	76	1	0.0800	0.7685
		2	0.0799	0.7685
		3	0.0791	0.7651
		4	0.0800	0.7650
		5	0.0814	0.7660
		Av. Performance	0.0801 (0.0008)	0.7666 (0.0018)
3:20:1	101	1	0.0773	0.7751
		2	0.0737	0.7833
		3	0.0778	0.7757
		4	0.0769	0.7775
		5	0.0778	0.7749
		Av. Performance	0.0767 (0.0017)	0.7773 (0.0035)
3:25:1	126	1	0.0788	0.7727
		2	0.0794	0.7722
		3	0.0775	0.7767
		4	0.0775	0.7757
		5	0.0775	0.7763
		Av. Performance	0.0781 (0.0009)	0.7747 (0.0021)
3:30:1	151	1	0.0774	0.7755
		2	0.0771	0.7786
		3	0.0781	0.7770
		4	0.0766	0.7820
		5	0.0787	0.7766
		Av. Performance	0.0776 (0.0008)	0.7780 (0.0025)

Table 2 (*ctd.*)

Neural net model	*Number of parameters*	*Trial*[a]	*Performance measures*[b] *ARV*	R^2
3:35:1	176	1	0.0847	0.7613
		2	0.0808	0.7712
		3	0.0811	0.7652
		4	0.0820	0.7666
		5	0.0794	0.7690
		Av. Performance	0.0816 (0.0020)	0.7667 (0.0038)
3:40:1	201	1	0.0766	0.7839
		2	0.0783	0.7763
		3	0.0764	0.7779
		4	0.0795	0.7740
		5	0.0798	0.7739
		Av. Performance	0.0781 (0.0016)	0.7772 (0.0041)
3:40:1	201	1	0.0766	0.7839
		2	0.0833	0.6377
		3	0.0867	0.6227
		4	0.0852	0.6299
		5	0.0869	0.6224
		Av. Performance	0.0858 (0.0016)	0.6270 (0.0068)
3:50:1	251	1	0.0868	0.6420
		2	0.0866	0.6532
		3	0.0869	0.6365
		4	0.0868	0.6421
		5	0.0870	0.6310
		Av. Performance	0.0868 (0.0002)	0.6400 (0.0082)
Conventional model	3	1	0.0880	0.6803
		2	0.0813	0.6080
		3	0.0855	0.6729
		4	0.0848	0.6930
		5	0.0870	0.6732
		Av. Performance	0.0863 (0.0032)	0.6768 (0.0080)

[a] The trials differ in initial conditions as well as in the random sequence of the input signals; epoch size of 20 patterns presented in random order; the training set consists of 496 patterns (50 percent of the available data).

[b] *ARV* defined in (13); R^2 defined in (14); average performance is the mean over the given performance values; the training set consists of 496 points, conventional model is the gravity model in (4). Standard deviation shown in parentheses.

The second performance measure used in this study is the coefficient of determination $R^2(S)$, a widely used goodness-of-fit measure in spatial interaction modelling (Fotheringham and O'Kelly 1989). This measure is defined as

$$R^2(S) = \frac{\sum_{l \in S} (\hat{y}_l - \bar{y})^2}{\sum_{l \in S} (y_l - \bar{y})^2} = \frac{1}{\hat{\sigma}^2} \frac{1}{N_S} \sum_{l \in S} (\hat{y}_l - \bar{y})^2 \tag{14}$$

with $0 \leq R^2 \leq 1$ and $\bar{y}$ denoting the average over the 20 predicted values in *S*.

The resulting fits of the model candidates, in terms of *ARV* and R^2, are reported in Table 2 for each of the five trials. Compared to the benchmark model's results, we find strikingly good fits for seven model candidates: (3:10:1), (3:15:1), (3:20:1), (3:25:1), (3:30:1), (3:35:1) and (3:40:1). The best average R^2 performance is achieved by the (3:30:1) model with $R^2 = 0.7780$ followed by (3:20:1) with 0.7773 and (3:40:1) with 0.7772. In terms of the average *ARV* performance a slightly different ranking is obtained. (3:20:1) performs slightly better than the (3:30:1) model, followed by the (3:40:1) and (3:25:1) models. The variation in *ARV* and R^2 performance due to initial conditions is very moderate.

It is well known, that there can be little connection between the training error, restricted to the training set, and the network's ability to generalise outside of that set. The (3:30:1) model unequivocally shows a better generalisation capability than the (3:20:1) model. The (3:30:1) network outperforms the (3:20:1) network on the testing set, in terms of the average *ARV* performance (0.4131 compared to 0.4198) and the average R^2 performance (0.5935 compared to 0.5827). In addition, (3:30:1) tends to be less sensitive to initial conditions. Thus, (3:30:1) rather than (3:20:1) has been chosen as the appropriate model for the problem at hand.

3.3 Model Estimation and the Overfitting Problem

Whereas the stage of identification was concerned with identifying an appropriate model from the family $\mathcal{N}_{3,I,1}$, the stage of parameter estimation is devoted to the determination of the magnitude and sign of the parameters of the (3:30:1) network model. A serious problem in model estimation is the problem of overfitting which is particularly serious for noisy real world data of limited record length. As opposed to computer generated data, the noise level in any real world context is not known a priori.

If the network mapping F_w fits the training data exactly, the capability of the network to generalise, that is to generate an acceptable output when a novel input signal is applied to the network, will often be poor. This arises from the rapid oscillations in the network function that are generally needed to fit the noisy training data. To improve the generalisation capability of the network model, it is necessary for the network mapping to represent the underlying trends in the data, rather than fitting all of the fine details of the data set (Bishop 1991).

Several strategies for handling the overfitting problem have been devised (Hecht-Nielsen 1990, Bishop 1991, Weigend et al. 1991). One possibility is to take a subset of input vectors from the training data. The subset may be chosen randomly or by a more systematic elimination procedure. We used the random

method of cross-validating to detect when overfitting occurs. It is crucial to split the whole available data set of 992 patterns into three separate subsets: a training (estimation) set, a validation test set and a testing (prediction) set. While the training set is directly used for training the neural network model, the validation test set is used only for the evaluation process, i.e., for determining when to stop the training process. The testing or prediction set is strictly set apart and never used in the estimation stage. The training set-validation set pair consists of two thirds of the available data. One quarter of these data (i.e., 148 patterns) has been used for validation process. It is crucial that the validation test set is only utilised to detect a statistically proper stopping point of the training process of the (3:30:1) model.

Figure 6 shows the performance of the (3:30:1) neural network model as a function of training time in epochs with an epoch size of 20 patterns. The average relative variances are given for the training set, the validation set and the prediction set. The *ARV* error of the model measured using the training set continuously decreases and seems to level out after 5,000 epochs. This is what one expects for a model of $\mathcal{N}_{I_1, I_2, I_3}$. The validation test set error as shown in Figure 6 decreases first, after 1,500 epochs only at a moderate rate until 4,250 epochs, then slightly increases and tends to approach an asymptotic value. If we assume that the error curve of the (3.30:1) model tested against the entire infinite set of possible patterns would be approximately the same as that of the validation set curve (Hecht-Nielsen 1990), which is only a crudely correct assumption, then clearly we would like to stop training when this curves arrives at its minimum. The minimum is reached after 4,250 epochs. At this stopping point, P, the model is used for prediction. In the specific choice of a training set-validation set combination shown in Figure 6, the fitting of the noise of the training set occurs to have only a little effect on the error of the validation set, which is also reflected in the prediction set curve.

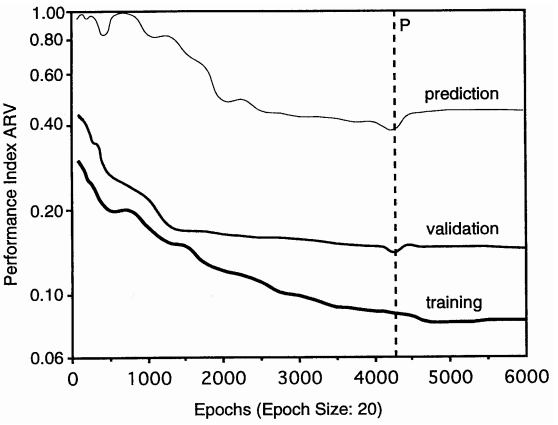


Figure 6 Training, validation and prediction set curves of the (3:30:1) network model as a function of training time in epochs (the vertical line P indicates the stopping point)

To get a feeling for the effect of the sampling error induced by using a specific training set-validation set combination, we analysed several such pairs. Because the telecommunication data set at hand is rather small, different pairs of training and validation test sets each, drawn randomly from the available data set, did lead to different *ARV* results differing by factors up to two. These variations are rather large compared to the small variations due to different random initial parameters. Consequently, the statistical procedure of cross-validating is somewhat unsatisfactory.

3.4 Neural Net and Gravity Model Predictions

The ultimate goal of the (3:30:1) neural net model is to predict interregional telecommunication traffic flows or in other words to determine how well the network learned to approximate the unknown input-output function for arbitrary values of $\boldsymbol{x}$. Thus, we briefly discuss the predictive quality of the neural net model and compare it with that of the gravity model. It is important to note that the network model is no longer allowed to adapt while it is being tested. To assess the prediction performance, we primarily use the average relative variance *ARV* as defined in Equation (13) and the coefficient of determination R^2 as defined in Equation (14).

Table 3 reports the prediction performance of the (3:30:1) network on the testing set in terms of *ARV* and R^2 averaged over the five trials. In order to make the comparison with the gravity model as close as possible, this model was estimated (tested) with the same data seen by the neural net model during training (testing). As can be seen by comparing the *ARV* and R^2 values, the neural network leads to a somewhat higher prediction performance in all trials. The average prediction quality measured in terms of *ARV* and R^2 is 0.4131 and 0.5932 respectively, compared to 0.4695 (*ARV*) and 0.5353 (R^2) for the gravity model. The prediction quality is rather stable over the different trials. In Table 4 the prediction accuracy achieved by the two alternative interregional teletraffic models is exemplified by a systematic sample of T_{rs} values.

Table 3 Testing performance of the (3:30:1) neural net and the gravity model[a]

	(3:30:1) Neural net model		*Conventional model*	
	ARV	R^2	*ARV*	R^2
Prediction (Testing)				
Trial 1	0.4063	0.5937	0.4630	0.5431
Trial 2	0.4057	0.5942	0.4611	0.5390
Trial 3	0.4063	0.5938	0.4582	0.5422
Trial 4	0.4077	0.5923	0.4761	0.5310
Trial 5	0.4064	0.5934	0.4892	0.5290
Average Performance	0.4131	0.5935	0.4695	0.5353
(Standard Deviation)	(0.0155)	(0.0007)	(0.0130)	(0.0068)

[a] The trials differ in initial conditions as well as in the random sequence of the input signals; *ARV* defined in Equation(13); R^2 defined in Equation (14); average performance is the mean over the given performance values; the testing set consists of 348 points, conventional model is the gravity model in Equation (4)

Table 4 Prediction accuracy of the (3:30:1) neural net and the gravity model: Some selected results

T_{rs}	*Observation*	*Model predictions*	
		Neural net	*Conventional*
$T_{2.17}$	69.7279	44.2051	57.8402
$T_{3.30}$	0.2113	0.1219	0.7010
$T_{4.7}$	12.1801	11.5071	13.8977
$T_{5.21}$	1.7500	1.6858	2.1387
$T_{11.23}$	0.4314	0.4695	0.5587
$T_{13.7}$	1.4137	1.8286	2.3708
$T_{16.9}$	10.4940	10.8594	19.4737
$T_{18.4}$	21.9815	20.5638	23.3007
$T_{20.11}$	4.4647	3.7864	3.4129
$T_{28.16}$	1.5310	1.6492	1.6535
$T_{29.18}$	15.0820	19.7773	30.9833

One means of further investigating the predictive power is the use of residual analysis. Figure 7 graphically displays in terms of

(a) the absolute residuals of the individual flows $(T_{rs}^{obs} - T_{rs}^{conv})$ compared to $(T_{rs}^{obs} - T_{rs}^{neur})$, and

(b) the relative residuals of the individual flows $(T_{rs}^{obs} - T_{rs}^{neur})/T_{rs}^{obs}$ and $(T_{rs}^{obs} - T_{rs}^{conv})/T_{rs}^{obs}$,

where both absolute and relative residuals are ordered by the size of the T_{rs}^{obs} flows. The main conclusions from this analysis can be summarised as follows:

(i) First, both models show a tendency to underpredict larger flows, especially those between dominant (provincial capital) regions. Examples include flows from Vienna to Klagenfurt, Salzburg to Linz, and Salzburg to Klagenfurt. They also underpredict flows involving dominant regions and neighbouring minor regions, such as, for example, the flows from Graz to Hartberg, Innsbruck to Reutte, Ried/Innkreis to Salzburg, and Wolfsberg to Klagenfurt. The neural network model underpredicts 63 out of 87 flows in the largest quartile, compared to 51 gravity model underpredictions.

(ii) Second, and in addition, the two alternate models share a tendency to overpredict smaller flows representing generally flows between minor regions further apart from each other. This is evidenced by 67 neural network and 75 gravity model overpredictions in the smallest quartile.

(iii) Third, the neural network model and the conventional model show a relatively similar pattern of residuals. Despite this similarity, however, the neural network

produces more accurate predictions for 188 of the 348 flows considered in the testing stage. This is also indicated by the average (that is, root-mean-squared) error of 20.52 percent, while the corresponding figure amounts to 24.56 percent for the gravity model. Many of the differences are small. But in some cases, the flows are replicated substantially more accurately by the neural network model. Figure 7(b2) provides evidence that flow errors of the gravity model may occasionally exceed even over 100 percent (four cases).

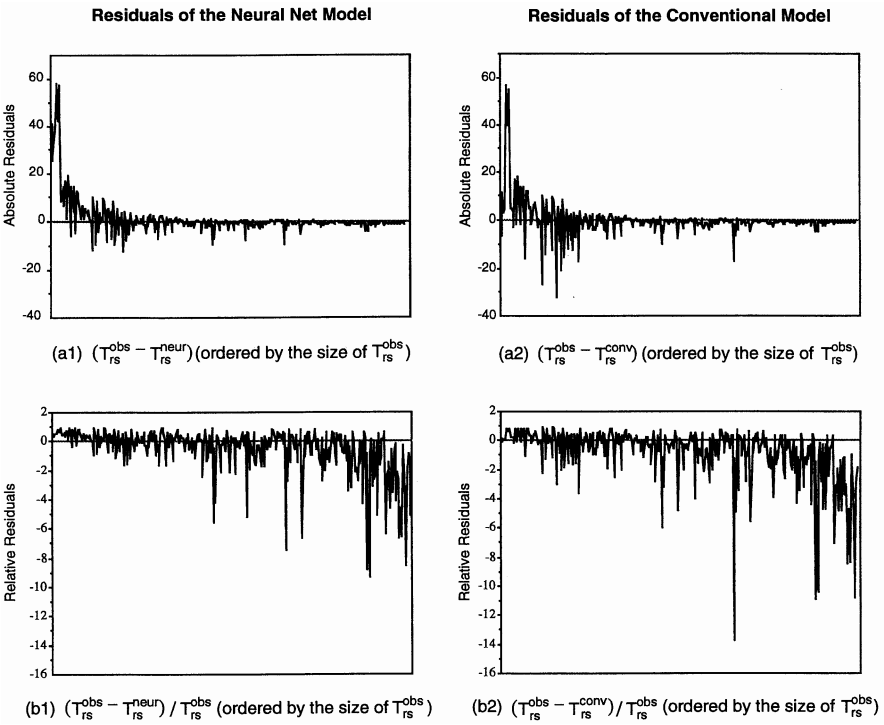


Figure 7 Residuals of (a) the 3:30:1 neural net and (b) the gravity model predictions

In summary, the analysis unequivocally shows that the neural net model outperforms the gravity model in terms of both the *ARV* and R^2 prediction performance as well as prediction accuracy, but to a lesser degree than previously expected. One reason for this might be that the validation set was relatively small and the statistical method of cross-validating the network against validation data did not indicate unequivocally the stopping point, another reason being the training procedure. In addition, there is an indication that memorisation of the training set is interfering with the ability of the (3:30:1) network to generalise. This is an issue for further research.

4 Summary and Conclusions

The primary advantage of the general neural network model set out in this paper over the classical regression approach to spatial interaction modelling lies in the fact that it has a more general functional form than the gravity model can effectively deal with. The neural network model implements a functional input-output relationship that is expressed in terms of a general, modifiable form. This functional form is modified via the adaptive setting of weights by means of the application of the feedbackpropagating technique and the additive (mean squared) error function to fit the specific mapping which is approximated. It may be viewed as a nonlinear regression function of a quite specific form (White 1989). The methods of nonlinear regression analysis evidently resemble those of neural network modelling. But none, if any, individual statistical regression function procedures have been developed as the neural network model presented in this paper.

For sigmoid processing units, relatively good local solutions were obtained for the interregional (3:30:1) neural net telecommunication model in all trials with different initial random network parameters. Sensitivity to initial conditions was rather moderate. This might have been mainly due to choosing relatively small initial parameter values, a small learning rate parameter η, a relatively large momentum rate parameter γ, a relatively large hidden layer and the logistic rather than the hyperbolic tangent transfer function for the problem at hand. On the noisy real world telecommunication data of limited record length the analysis illustrated the superiority of the neural network model over the classical regression approach. But neural network modelling requires a procedure to deal with the problem of overfitting. The statistical method of cross-validating the network against a reserved set of validation data is certainly unsatisfactory at least due to two reasons. First, the results depend on the specific training set-validation set pair used for deciding when to stop training. Second, it is not always completely clear from the *ARV* of the validation set when the training process should be stopped to avoid overfitting. The issue of overfitting deserves further research efforts in future. One alternative strategy to tackle this problem might be the strategy of weight elimination suggested by Weigend et al. (1991). This strategy involves the extension of the gradient method by adding a complexity term to the error (cost) function which associates a cost element with each connection.

Neural networks have an important role to play in geography and regional science not only in the application domain of spatial interaction modelling, but also in exploratory spatial data analysis. They can principally be used to cope with problems such as very large volumes of spatial data, missing data, noisy data and fuzzy information for which conventional statistical techniques may be inappropriate or cumbersome to use. It is hoped that the methodology outlined in this paper will make the study of neural networks more accessible to the regional science community and will stimulate further research in this new and promising field.

***Acknowledgments*:** The quality of this paper was substantially improved by the comments provided by three anonymous reviewers on the first submission of the paper, we gratefully appreciate their comments. The second author wishes to acknowledge the assistance and computational environment provided when she was a visiting professor at the Vienna University of Economics and Business Administration, Department of Economic and Social Geography. The authors are grateful for the funding provided by the Austrian Fonds zur Förderung der Wissenschaftlichen Forschung (P0992-TEC) and the U.S. National Science Foundation (SBR-9300633). Moreover, both authors would like to thank Petra Staufer for her valuable assistance in the preparation of the manuscript and illustrations.

References

Amari S.-I. (1990): Mathematical foundations of neurocomputing. In: *Proceedings of the IEEE* 78, pp. 1443-1463

Barron A.R. (1993): Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory* 39, 930 -945

Bishop C.M. (1991): Improving the generalisation properties of radial basis function neural networks, *Neural Computation* 3 (4), 579-588

Cybenko G. (1989): Approximation by superpositions of sigmoidal function, *Mathematics of Control, Signals, and Systems* 2 (4), 303-314

Fischer M.M. (1993): Travel demand. In: Polak J. and Heertje A. (eds.) *European Transportation Economics*, Basil Blackwell, Oxford [UK] and Cambridge [MA], pp. 6-32

Fischer M.M. and Gopal S. (1993): Neurocomputing – a new paradigm for geographic information processing, *Environment and Planning A* 25 (6), 757-760

Fischer M.M., Essletzbichler J., Gassler H. and Trichtl G. (1993): Interregional and international telephone communication. Aggregate traffic models and empirical evidence for Austria, *Sistemi Urbani* 15 (2/3), 121-135

Fotheringham S.A., and O'Kelly M.E. (1989): *Spatial Interaction Models*: *Formulations and Applications*, Kluwer Academic Publishers, Dordrecht, Boston, London

Funahashi K. (1989): On the approximate realization of continuous mappings by neural networks, *Neural Networks* 2, 183-192

Gillespie A. and Howard W. (1988): Telecommunications and the reconstruction of regional comparative advantage, *Environment and Planning A* 20 (10), 1311-1321

Guldmann J.-M. (1992): Modelling residential and business telecommunication flows: A regional point-to-point approach, *Geographical Analysis* 24, 121-141

Gyer M.S. (1992): Adjuncts and alternatives to neural networks for supervised classification, *IEEE Transactions on Systems, Man, and Cybernetics* 22 , 35-47

Hecht-Nielsen R. (1990): *Neurocomputing*, Addison-Wesley, Reading [MA]

Hornik K.M., Stinchcombe M. and White H. (1989): Multi-layer feedforward networks are universal approximators, *Neural Networks* 2, 359-366

Levin E., Tishby N. and Solla S.A. (1990): A statistical approach to learning and generalisation in layered neural networks. In: *Proceedings of the IEEE* 78, pp. 1568-1575

Openshaw S. (1992): Modelling spatial interaction using a neural net. In: Fischer M.M. and Nijkamp P. (eds.) *Geographic Information Systems, Spatial Modelling, and Policy Evaluation***,** Springer, Berlin, Heidelberg, New York, pp. 147-164

Pacey P.L. (1983): Long distance demand: A point-to-point model, *Southern Economic Journal* 49, 1094-1107

Rietveld P. and Janssen L. (1990): Telephone calls and communication barriers. The case of the Netherlands, *The Annals of Regional Science* 24, 307-318

Rossera F. (1990): Discontinuities and barriers in communications. The case of Swiss communities of different language, *The Annals of Regional Science* 24, 319-336

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

Salomon I. (1986): Telecommunications and travel relationships: A review, *Transportation Research* 3, 223-238

Shah S., Palmieri F. and Datum M. (1992): Optimal filtering algorithms for fast learning in feedforward neural networks, *Neural Networks* 5, 779-787

Weigend A.S., Rumelhart D.E. and Huberman B.A. (1991): Backpropagation, weight-elimination and time series prediction. In: Touretzky D.S., Elman J.L., Sejnowski T.J. and Hinton G.E. (eds.) *Connectionist Models. Proceedings of the 1990 Summer School*, Morgan Kaufmann Publishers, San Mateo [CA], pp.105-116

White H. (1989): Some asymptotic results for learning in single hidden layer feedforward network models, *Journal of American Statistical Association* 84, 1003-1013

Widrow B. and Lehr M.A. (1990): 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. In: *Proceedings of the IEEE* 78, pp. 1415-1447

Appendix: *Parameters of the (3:30:1)-Interregional Teletraffic Neural Network Model*

The (3:30:1) network was trained for 4,250 epochs with a learning rate $\eta = 0.15$ and a momentum $\gamma = 0.8$. The weights were updated after each 20 patterns presented in random order. The connection weights and biases of the network are given below in Table A1. When simulated serially on a SparcStation 1+, the training of the 3:30:1 takes less than 6 CPU-minutes for 4,250 epochs. Once the network parameters have been determined, predictions are extremely fast. The parameter estimates of the gravity model (4) are explicitly given in Table A2.

Table A1 Parameter estimates $W_{1, i_1, i_2}; W_{2, i_2, i_3}$ of the (3:30:1) neural net interregional teletraffic model (Trial 3 see Table 3)

Weights W_{1, i_1, i_2}	*from input unit* $i_1 = 1$		*Input unit* $i_1 = 2$		*Input unit* $i_1 = 3$		*Bias unit*	
	Start	*Final*	*Start*	*Final*	*Start*	*Final*	*Start*	*Final*
to hidden unit 1	0.0073	0.1045	0.0160	0.1022	0.0662	–0.0839	0.0680	–0.0256
to hidden unit 2	–0.0985	–0.5126	–0.0213	–0.3944	–0.0116	0.2485	0.0164	–0.0347
to hidden unit 3	0.0022	–0.2100	–0.0488	–0.2391	–0.0134	0.0745	0.0829	0.0070
to hidden unit 4	–0.0150	–0.1397	0.0981	–0.0181	–0.0156	0.0045	–0.0787	–0.1552
to hidden unit 5	0.0005	–0.1220	–0.0002	–0.1118	–0.0246	–0.0011	–0.0719	–0.1457
to hidden unit 6	–0.0818	–0.1307	0.0509	0.0048	–0.0890	–0.1207	–0.0439	–0.1206
to hidden unit 7	0.0504	0.4725	–0.0079	0.3707	–0.0081	–0.4110	0.0757	–0.0372
to hidden unit 8	–0.0064	–0.3705	–0.0871	–0.4116	0.0868	0.2903	0.0823	0.0098
to hidden unit 9	0.0225	–0.0459	0.0878	0.0233	0.0781	0.0511	–0.0063	–0.0916
to hidden unit 10	0.0251	0.2904	0.0423	0.2780	0.0656	–0.2065	–0.074	–0.1678
to hidden unit 11	–0.0201	0.3393	0.0859	0.4052	0.0406	–0.3144	0.0581	–0.0525
to hidden unit 12	0.0269	0.3442	0.0993	0.3788	0.0072	–0.3166	0.0331	–0.0728

Table A1 (*ctd.*)

Weights W_{1,i_1,i_2} *from input unit* $i_1 = 1$			*Input unit* $i_1 = 2$		*Input unit* $i_1 = 3$		*Bias unit*	
	Start	*Final*	*Start*	*Final*	*Start*	*Final*	*Start*	*Final*
to hidden unit 13	–0.0265	0.4407	0.0603	0.4770	–0.0953	–0.5211	0.0510	–0.0506
to hidden unit 14	0.0729	0.0205	0.0454	–0.0049	0.0812	0.0398	0.0411	–0.0472
to hidden unit 15	0.0935	0.7405	0.0783	0.6618	0.0382	–0.5259	–0.0865	–0.1940
to hidden unit 16	0.0536	0.2845	0.0461	0.2475	–0.0826	–0.3396	0.0754	–0.0235
to hidden unit 17	0.0714	0.4191	0.0007	0.3104	–0.0193	–0.3629	0.0441	–0.0606
to hidden unit 18	–0.0288	0.1483	–0.0149	0.1440	0.0210	–0.1846	0.0520	–0.0414
to hidden unit 19	–0.0737	–0.1998	0.0151	–0.0986	–0.0486	–0.0233	0.0495	–0.0276
to hidden unit 20	0.0780	–0.0512	–0.0344	–0.1507	0.0945	0.1178	–0.0229	–0.1031
to hidden unit 21	–0.0014	0.3241	0.0135	0.3032	–0.0559	–0.3749	0.0369	–0.0601
to hidden unit 22	–0.0408	–0.0891	–0.0593	–0.1013	–0.0236	–0.0540	0.0182	–0.0607
to hidden unit 23	–0.0835	–0.4889	–0.0225	–0.3865	0.0454	0.2853	0.0784	0.0126
to hidden unit 24	–0.0288	0.2170	–0.0195	0.1998	–0.0629	–0.3207	0.0732	–0.0214
to hidden unit 25	0.0596	0.1735	–0.0745	0.0264	–0.0183	–0.1775	0.0304	–0.0589
to hidden unit 26	–0.0787	–0.4814	–0.0585	–0.4193	0.0566	0.3126	–0.0502	–0.0976
to hidden unit 27	–0.0383	–0.1429	–0.0089	–0.1026	–0.0031	0.0053	0.0366	–0.0425
to hidden unit 28	–0.0138	–0.2159	0.0405	–0.1424	0.0372	0.1190	–0.0453	–0.1182
to hidden unit 29	0.0619	–0.1300	–0.0393	–0.2139	–0.0700	0.0060	–0.0849	–0.1535
to hidden unit 30	–0.0251	0.1863	0.0369	0.2245	0.0070	–0.2300	0.0805	–0.0181

Weights W_{2,i_2,i_3} *to output unit* $i_3 = 1$		
	Start	*Final*
from hidden unit 1	–0.0590	–0.2391
from hidden unit 2	0.0410	0.1182
from hidden unit 3	–0.0448	–0.7077
from hidden unit 4	–0.0297	–0.3492
from hidden unit 5	–0.0787	–0.1551
from hidden unit 6	–0.0362	–0.1870
from hidden unit 7	–0.0206	–0.0704
from hidden unit 8	0.0929	0.6580
from hidden unit 9	–0.0103	–0.6427
from hidden unit 10	–0.0466	–0.0925
from hidden unit 11	0.0603	0.3939
from hidden unit 12	0.0802	0.5462
from hidden unit 13	0.0234	0.5274
from hidden unit 14	0.0795	0.7475
from hidden unit 15	–0.0460	–0.0629
from hidden unit 16	0.0997	10.0524
from hidden unit 17	–0.0166	0.4209
from hidden unit 18	0.0517	0.5621
from hidden unit 19	0.0834	0.2221
from hidden unit 20	–0.0118	–0.2163
from hidden unit 21	–0.0212	–0.2186
from hidden unit 22	0.0658	0.5064
from hidden unit 23	0.0395	–0.1266
from hidden unit 24	–0.0286	–0.7038
from hidden unit 25	0.0755	0.3575
from hidden unit 26	0.0365	0.1629

Table A1 (*ctd.*)

Weights W_{2,i_2,i_3} *to output unit* $i_3 = 1$	*Start*	*Final*
from hidden unit 27	–0.0098	–0.7256
from hidden unit 28	0.0029	–0.1940
from hidden unit 29	–0.0502	–0.3205
from hidden unit 30	–0.0834	–0.2587
from bias unit $i_2 = 31$	–0.0590	–0.2357
R^2	0.1022	0.7161
R^2 adjusted	0.1021	0.7159

Table A2 Parameter estimates of the gravity model

	Gravity model (4)	
Constant	–19.5976	
$\hat{\alpha}_1$	0.7297	(52.410)[a]
$\hat{\alpha}_2$	0.7035	(49.366)[a]
$\hat{\alpha}_3$	–0.7156	(–31.650)[a]
R^2 (adjusted)	0.5390	(0.5387)

[a] *t*-values

8 A Genetic-Algorithms Based Evolutionary Computational Neural Network for Modelling Spatial Interaction Data

with *Yee Leung*

Building a feedforward computational neural network model (*CNN*) *involves two distinct tasks: determination of the network topology and weight estimation. The specification of a problem adequate network topology is a key issue and the primary focus of this contribution. Up to now, this issue has been either completely neglected in spatial application domains, or tackled by search heuristics* (*see Fischer and Gopal 1994*). *With the view of modelling interactions over geographic space, the current chapter considers this problem as a global optimisation problem and proposes a novel approach that embeds backpropagation learning into the evolutionary paradigm of genetic algorithms. This is accomplished by interweaving a genetic search for finding an optimal CNN topology with gradient-based backpropagation learning for determining the network parameters. Thus, the model builder will be relieved of the burden of identifying appropriate CNN-topologies that will allow a problem to be solved with simple, but powerful learning mechanisms, such as backpropagation of gradient descent errors. The approach has been applied to the family of three inputs, single hidden layer, single output feedforward CNN models using interregional telecommunication traffic data for Austria, to illustrate its performance and to evaluate its robustness.*

1 Introduction

The recent emergence of computational intelligence technologies such as artificial life, evolutionary computation and neural networks has been accomplished by a virtual explosion of research, spanning a range of disciplines, perhaps wider than any other contemporary intellectual endeavour. Researchers from such diverse fields such as neuroscience, computer science, cognitive science, physics, engineering, statistics, mathematics, computational economics and GeoComputation are daily making substantial contributions to the understanding, development and applications of computational adaptive systems.

With a few exceptions (notably Openshaw 1988, 1993, 1998, Leung 1994, 1997, Fischer 1998, Fischer et al. 1997, Fischer and Gopal 1994, Gopal and Fischer 1996, Openshaw and Openshaw 1997, Nijkamp et al. 1996) geographers and regional scientists have been rather slow in realising the potential of these

novel technologies for spatial modelling. Recently, neural spatial interaction models with three inputs and a single output have been established as a powerful class of universal function approximators for spatial interaction flow data (see Fischer and Gopal 1994).

One of the open issues in neural spatial interaction modelling includes the model choice problem, also termed the problem of determining an appropriate network topology. It consists of optimising the complexity of the neural network model in order to achieve the best generalisation. Considerable insight into this phenomenon can be obtained by introducing the concept of the bias-variance trade-off, in which the generalisation error is disaggregated into the sum of the squared bias plus the variance. A model that is too simple, or too inflexibe, will have a large bias, while one that has too much flexibility in relation to the particular data set will have a large variance. The best generalisation is obtained when the best compromise between the conflicting requirements of small bias and small variance are achieved. In order to find the optimal balance between the bias and the variance it is necessary to control the effective complexity of the model, complexity measured in terms of the number of adaptive parameters (Bishop 1995).

Various techniques have been developed in the neural network literature to control the effective complexity of neural network models, in most cases as part of the network training process itself. The most widely used approach is to train a set of model candidates and choose that one which gives the best value for a generalisation performance criterion. This approach requires significant computational effort and yet it only searches a restricted class of models. An obvious drawback of such an approach is its trial and error nature. An alternative and a more principled approach to the problem utilised by Fischer et al. (1997) is to start with an oversized model and gradually remove either parameter weights or complete processing units in order to arrive at a suitable model. This technique is known as *pruning technique*. One difficulty with such a technique is associated with the threshold definitions that are used to decide which adaptive parameters or processing units are important.

Yet another way, to optimise the model complexity for a given training data set is the procedure of *stopped* or *cross-validation* that had been used by Fischer and Gopal (1994). Here, an overparameterised model is trained until the error on further independant data, called validation data set, deteriorates, then training is stopped. This contrasts to the other above approaches since model choice does not require convergence of the training process. The training process is used to perform a directed search of the weight space for a model that does not overfit the data and, thus, demonstrates generalisation performance. This approach has its shortcomings too. First, it might be hard in practice to identify when to stop training. Second, the results may depend on the specific training set-validation set pair chosen. Third, the model which has the best performance on the validation set might not be the one with the best performance on the test set.

Though these approaches address the problem of neural network model choice, they investigate only restricted topological subsets rather than the complete class of computational neural network (CNN) architectures. As a consequence, these techniques tend to force a task into an assumed architectural class rather than

fitting an appropriate architecture to the task. In order to circumvent this deficiency, we suggest genetic algorithms, a rich class of stochastic global search methods, for determining optimal network topologies. Genetic search on the space of CNN topologies relieves the model builder of the burden of identifying the network structure (topology) that would otherwise have to be done by hand using trial and error. Standard genetic algorithms, with no tricks to speed up convergence, are very robust and effective for global search, but very slow in fine-tuning (i.e. converging) a good solution once a promising region of the search space has been identified (Maniezzo 1994). This motivates one to marry the advantages of genetic evolution and gradient-based (local) learning. Genetic algorithms can be used to provide a model of evolution of the topology of CNNs, and supervised learning may be utilised to provide simple, but powerful learning mechanisms. Backpropagation learning appears to be a natural local search integration for genetic evolution, in the case of CNN optimisation.

The remainder of this paper is organised as follows. Section 2 describes the basic features of neural spatial interaction models along with gradient-based backpropagation learning as standard approach to parameter estimation. Section 3 introduces the fundamentals of genetic algorithms and concludes with a brief overview of how they can be applied to network modelling. Section 4 presents the hybrid system, called GENNET (standing for GENetic evolution of computational neural NETworks) that interweaves a genetic search for an appropriate network topology (in the space of CNN topologies) with gradient-based backpropagation learning (in the weight space) for determining the network parameters. Modelling spatial interaction data has special significance in the historical development of mathematical modelling in geography and regional science, the testing ground for new approaches. The testbed for the evaluation uses interregional telecommunication traffic data from Austria because they are known to pose a difficult problem to neural networks using backpropagation learning due to multiple local minima and there is a CNN benchmark available (see Fischer and Gopal 1994, Gopal and Fischer 1996). Section 5 reports on a set of experimental tests carried out to identify an optimal parameter setting and to evaluate the robustness of the approach suggested with respect to its parameters, using a measure which provides an appropriate compromise between network complexity and in-sample and out-of-sample performances. Section 6 summarises the results achieved, and outlines directions for future research.

2 Neural Spatial Interaction Models

Neural spatial interaction models are termed *neural* in the sense that they have been inspired by neuroscience. But they are more closely related to conventional spatial interaction models of the gravity type than they are to neurobiological models. They are special cases of general feedforward neural network models. Rigorous mathematical proofs for the universality of such models employing continuous sigmoid type transfer functions (see among others Hornik et al. 1989)

establish the three input single output single hidden layer neural spatial interaction models developed by Fischer and Gopal (1994) as a powerful class of universal approximators for spatial interaction flow data.

Such models may be viewed as a particular type of an input-output model. Given a three-dimensional input vector $\boldsymbol{x}$ that represents measures of origin propulsiveness, destination attractiveness and spatial separation, the neural model produces a one-dimensional output vector $\boldsymbol{y}$, say

$$\boldsymbol{y} = \Phi(\boldsymbol{x}, \boldsymbol{w}) = \psi\left(\sum_{j=0}^{J} \beta_j \; \varphi_j\left(\sum_{n=0}^{3} \alpha_{jn} \; x_n\right)\right) \tag{1}$$

representing spatial interaction flows from regions of origin to regions of destination. J denotes the number of hidden units, $\varphi_j(\cdot)$ (j = 1, ..., J) and $\psi(\cdot)$ are transfer (activation) functions of, respectively, the jth hidden and the output unit. The symbol $\boldsymbol{w}$ represents a $(5J + 1)$-dimensional vector of all the α- and β-network weights (parameters). x_0 represents a bias signal equal to 1. The transfer functions $\varphi_j(\cdot)$ and $\psi(\cdot)$ are assumed to be differentiable, nonlinear; moreover, $\varphi_j(\cdot)$ is generally, but not necessarily assumed to be identical for j = 1, ..., J.

Each neural spatial interaction model $\Phi(\boldsymbol{x}, \boldsymbol{w})$ can be represented in terms of a network diagram (see Figure 1) such that there is a one-to-one correspondence between components of Φ and the elements of the diagram. Equally, any topology of a three layer network diagram with three inputs and a single output, provided it is feedforward, can be translated into the corresponding neural spatial interaction model. We can, thus, consider model choice in terms of topology selection (i.e., choice of the number of hidden units) and specification of the transfer functions ψ and φ_j (j = 1, ..., J).

When approximating the analytically unknown input-output function $\mathcal{F}$: $\Re^3 \rightarrow \Re$ from available samples $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ with $\mathcal{F}(\boldsymbol{x}^k) = \boldsymbol{y}^k$, we have to determine the structure (i.e., the choice of ψ and φ_j with j = 1, ..., J, and the network topology) of the spatial interaction model Φ first, and from that finding an optimal set $\boldsymbol{w}^{opt}$ of adaptive parameters. Obviously, these two processes are intertwined. If a good set of transfer functions can be found, the success of which depends on the particular real world problem, then the task of weight learning (parameter estimation) generally becomes easier to perform.

In all the models under investigation, the hidden unit and output unit transfer functions [φ_j with j = 1, ..., J and ψ] are chosen to be identical and the logistic function. This specification of the general model class Φ leads to neural spatial interaction models, say Φ^L, of the following type

$$y = \Phi^L(\boldsymbol{x}, \boldsymbol{w}) = \left\{1 + \exp\left[-\lambda \sum_{j=0}^{J} \beta_j \left[1 + \exp\left(-\lambda \sum_{n=0}^{3} \alpha_{jn} \, x_n\right)\right]^{-1}\right]\right\}^{-1} \tag{2}$$

with values λ close to unity. Thus, the problem of determining the model structure is reduced to determine the network topology of the model (i.e., the number J of hidden units). Hornik et al. (1989) have demonstrated with rigorous mathematical proofs that network output functions such as Φ^L can provide an accurate approximation to any function $\mathcal{F}$ likely to be encountered, provided that J is sufficiently large. This universal approximation property establishes the attractivity of the spatial interaction models considered in this contribution.

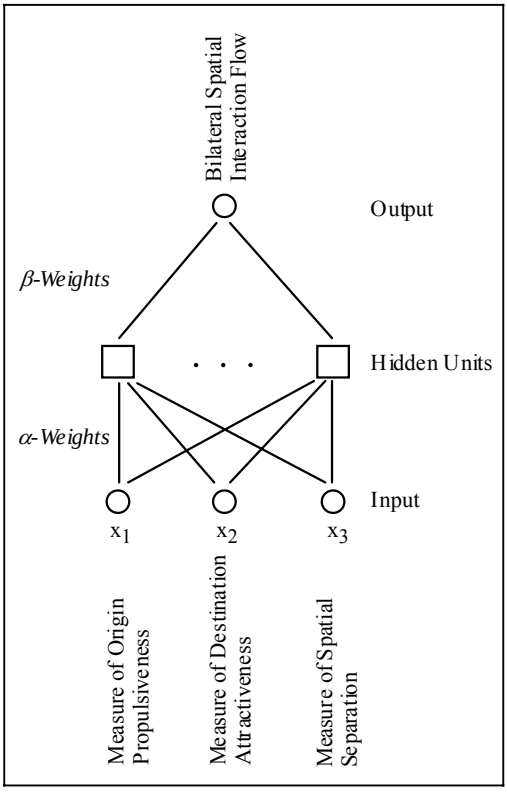


Figure 1 Representation of the general class of neural spatial interaction models as defined by Equation (1) [biases not shown]

Without loss of generality, we assume Φ^L to have a fixed topology, i.e. J is predetermined. Then, the role of learning is to find suitable values for network weights $\boldsymbol{w}$ of this model such that the underlying input-output relationship $\mathcal{F}$: $\Re^3 \rightarrow \Re$ represented by the training set $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ $k = 1, 2, ...;$ i.e., $\mathcal{F}(\boldsymbol{x}^k) = \boldsymbol{y}^k$, is approximated or learned, where k indexes the training instance. $\boldsymbol{y}^k$ is a 1-dimensional vector representing the desired network output (i.e. the spatial in-

teraction flows) upon presentation of $\boldsymbol{x}^k$ (i.e. measures of propulsiveness, destination attractiveness and spatial separation). Since the learning here is supervised (i.e., target outputs $\boldsymbol{y}^k$ are available), an error (objective, performance) function may be defined to measure the degree of approximation for any given setting of the network's weights. A commonly used, but by no means the only error function is the least squares criterion which is defined for on-line learning as follows

$$E(\boldsymbol{w}) = \tfrac{1}{2} \sum_{(\boldsymbol{x}^k, \boldsymbol{y}^k)} (\boldsymbol{y}^k - \Phi(\boldsymbol{x}^k, \boldsymbol{w}))^2. \tag{3}$$

Once a suitable error function is formulated, learning can be viewed as an optimisation process. That is, the error function serves as a criterion function, and the learning algorithm seeks to minimise the criterion function such as (3) over the space of possible weight settings. Using (3) an optimal parameter set $\boldsymbol{w}^{opt}$ may be chosen as:

$$\boldsymbol{w}^{opt} : E(\boldsymbol{w}^{opt}) = \min_{\boldsymbol{w}} E(\boldsymbol{w}). \tag{4}$$

The most prominent learning algorithm which has been proposed in the neural network literature to solve this minimisation problem is backpropagation (BP) learning (Rumelhart et al. 1986) combined with the gradient descent technique which allows for efficient updating of the parameters due to the feedforward architecture of the spatial interaction models.

In its standard version backpropagation learning starts with an initial set of random weights $\boldsymbol{w}_0$ and then updates them by

$$\boldsymbol{w}_\tau = \boldsymbol{w}_{\tau-1} + \eta \nabla \Phi(\boldsymbol{x}^k, \boldsymbol{w}_{\tau-1}) (\boldsymbol{y}^k - \Phi(\boldsymbol{x}^k, \boldsymbol{w}_{\tau-1})) \qquad k = 1, 2, ..., K, \tag{5}$$

where $\boldsymbol{w}$ is the $(5J + 1)$-dimensional vector of network weights to be learned; its current estimate at time $\tau - 1$ is denoted by $\boldsymbol{w}_{\tau-1}$; $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ is the training pattern presented at time τ; Φ is the network function; η is a fixed step size (the so-called learning rate), $\nabla\Phi$ is the gradient (the vector containing the first-order partial derivatives) of Φ with respect to the parameters $\boldsymbol{w}$. Note that parameters are adjusted in response to errors in hitting the target, $\boldsymbol{y}^k - \Phi(\boldsymbol{x}^k, \boldsymbol{w}_{\tau-1})$. The performance of backpropagation learning can be greatly influenced by the choice of η. Note that (5) is the parameter update equation of the on-line, rather than the batch version of the backpropagation learning algorithm. For very small η (i.e. approaching zero) on-line backpropagation learning approaches batch backpropagation (Finnoff 1993). But there is a non-negligible stochastic element (i.e. $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ are drawn at random) in the training process that gives on-line backpropagation a quasi-annealing character in which the cumulative gradient is continuously perturbed, allowing the search to escape local minima with small and shallow basins

of attraction (Hassoun 1995). Although many modifications of this procedure (notably the introduction of a momentum term, $\mu \Delta \boldsymbol{w}_{\tau-1}$, into the weight update equation and the use of a variable step size, denoted $\eta_{\tau-1}$) and alternative optimisation procedures have been suggested over the past few years, experience shows that surprising good network performance can often be achieved with this on-line (local) learning algorithm in real world applications (see, e.g., Fischer et al. 1997 for an epoch-based version).

As the optimum network size and topology (i.e. the number of hidden layers and hidden units, connectivity) are usually unknown, the search of this optimum requires a lot of networks to be trained on a trial and error basis. Moreover, there is no guarantee that the network obtained is globally optimal. In this contribution, we view this issue as a global optimisation problem and, therefore, suggest the application of genetic algorithms which provides multi-point global optimal search for the network topology.

3 Basics of the Canonical Genetic Algorithm

Genetic algorithms (GAs) are revealing to be a very rich class of stochastic search algorithms inspired by evolution. These techniques are population oriented and use selection and recombination operators to generate new sample points in a search space. This is in contrast to standard programming procedures that usually follow just one trajectory (deterministic or stochastic), perhaps repeated many times until a satisfactory solution is reached. In the GA approach, multiple stochastic solution trajectories proceed simultaneously, permitting various interactions among them towards one or more regions of the search space. Compared with single-trajectory methods, such as simulated annealing, a GA is intrinsically parallel and global. Local 'fitness' information from different members is mixed through various genetic operators, especially the crossover mechanism, and probabilistic soft decisions are made concerning removal and reproduction of existing members. In addition, GAs require only simple computations, such as additions, random number generations, and logical comparisons, with the only major burden that a large number of fitness function evaluations have to be performed (Qi and Palmieri 1994). This section will review the fundamentals of the canonical genetic algorithm as introduced by Holland (1975), and then show how genetic algorithms can be used as means to perform the task of model choice (topology optimisation) in the spatial interaction arena.

In its simplest form, the canonical genetic algorithm is used to tackle static discrete optimisation problems of the following form:

$$\max \{ f(\boldsymbol{s}) \mid \boldsymbol{s} \in \Omega \} \tag{6}$$

assuming that $0 < f(\boldsymbol{s}) < \infty$ for all $\boldsymbol{s} \in \Omega = \{0, 1\}^d$ and $f(\boldsymbol{s}) \neq \text{const}$. In this case, the objective function is called the *fitness function* f: $\Omega \rightarrow \Re$, and the d-dimen-

sional binary vectors in Ω are called *strings* (sometimes also genotypes or chromosomes). The size of the search space Ω is 2^d and forms a hypercube. The GA samples the corners of this d-dimensional hypercube. The P-tuple of individual strings $(\boldsymbol{s}_1, ..., \boldsymbol{s}_P)$ is said to be a *population* S. Each individual $\boldsymbol{s}_k \in S$ represents a feasible solution of problem (6) and its objective function value $f(\boldsymbol{s}_k)$ is said to be its *fitness* which is to be maximised. f is called the fitness function. If the problem is to minimise a given objective function g

$$\min \{g(\boldsymbol{x}) \mid \boldsymbol{x} \in \Sigma\} \tag{7}$$

assuming that $0 < g(\boldsymbol{x}) < \infty$ for all $\boldsymbol{x} \in \Sigma \subset \Re^d$ and $g(\boldsymbol{x}) \neq$ const, then it is necessary first, to map the 'real' search space Σ into the representation space Ω of binary (fixed-length) strings, and, second, to transform the objective function $g(\boldsymbol{x})$ into an appropriate fitness function $f(\boldsymbol{s})$ such that the maximisers of $f(\boldsymbol{s})$ correspond to the minimisers of $\boldsymbol{x}$. This situation is schematically illustrated in Figure 2.

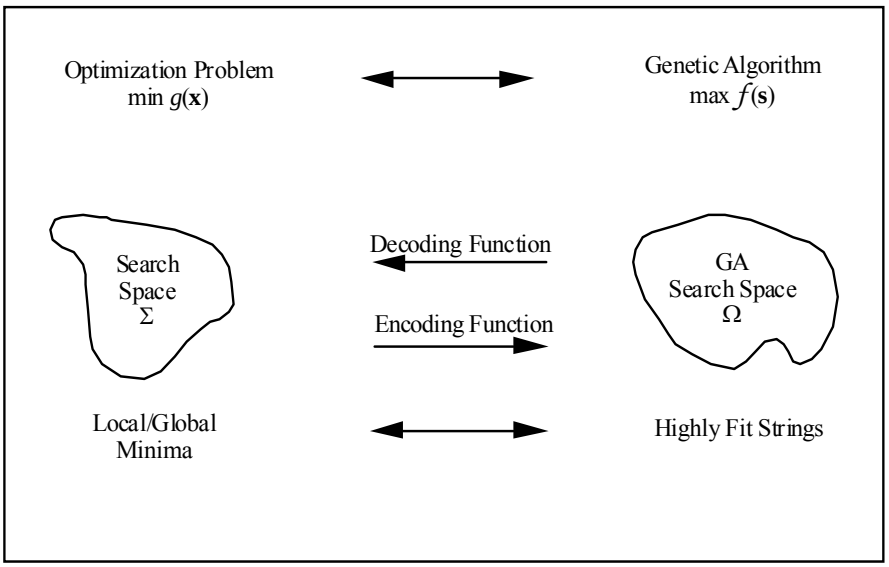


Figure 2 The dual representation scheme used in the GA-approach [see Hassoun 1995]

The standard genetic algorithm can be sketched as follows:

Step 1: Randomly generate an initial population of, say, P binary strings of length d : $S(0) = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_P\} \subset \Omega$. }.

Step 2: Compute the fitness score $f(\boldsymbol{s}_k)$ of each individual string $\boldsymbol{s}_k$ of the current population $S(t)$.

Step 3: Generate an intermediate population (termed mating pool) by applying the *selection* operator.

Step 4: Generate $S(t+1)$ by applying recombination operators (*crossover* and *mutation*) to the intermediate population.

Step 5: $t = t + 1$ and continue with *Step 2* until some stopping criterion applies (in this case designate the best-so-far individual as the result of the GA).

The first step generates an initial population $S(0)$, i.e. $S(0) = \{\boldsymbol{s}_1, ..., \boldsymbol{s}_P\} \subset \Omega$. In the canonical GA each member of $S(0)$ is a binary string of length d that corresponds to the problem coding. $S(0)$ is usually generated randomly, because it is not known a priori where the globally optimal strings in Ω are likely to be found. From this initial population, subsequent populations $S(1), ..., S(t), ...$ will be computed by employing the three genetic operators of selection (reproduction), crossover and mutation.

After calculating the relative fitness for all the strings in the current population $S(t)$ (*Step 2*), selection is carried out. In the canonical GA the *roulette wheel selection* (Goldberg 1989) technique is used for constructing the intermediate population (*Step 3*), i.e. a proportional selection technique, where the intermediate population is determined by P independent random experiments. The probability that individual $\boldsymbol{s}_k$ is selected from tuple $(\boldsymbol{s}_1, ..., \boldsymbol{s}_P)$ to be member of the intermediate population at each experiment is given by

$$p_r(\boldsymbol{s}_k \text{ is selected}) = \frac{f(\boldsymbol{s}_k)}{\sum_{k=1}^{P} f(\boldsymbol{s}_k)} > 0. \tag{8}$$

That is, strings in the current population are copied (i.e. duplicated) and placed in the intermediate population proportional to their fitness relative to other individuals in the population.

After selection has been carried out the construction of the intermediate population is complete. Then crossover and mutation are applied to the intermediate population to create the next population $S(t+1)$ (*Step 4*). Crossover and mutation provide a means of generating new sample points in Ω while partially preserving distribution of strings across hyperplanes which is observed in the intermediate population. *Crossover* is a recombination mechanism to explore new regions in Ω. The crossover operator is applied with some probability $p_c \in [0, 1]$. To apply the one-point crossover operator, for example, the individuals of the intermediate population are randomly paired. Each pair (*parents*) is then combined, choosing one point in accordance with a uniformly distributed probability over the length of the individual strings and cutting them in two parts accordingly. The two new strings, called *offspring*, are formed by the juxtaposition of the first part of one parent and the last part of the other parent. For example, Figure 3 illustrates a crossover for two 7-bit strings. In this case, the crossing site is 5, so the bits from the two strings are swapped after the fifth bit. It

should be noted that we can also perform multi-point crossover (Spears and De Jong 1991) and uniform crossover (Syswerda 1989) for evolution.

(a)	(b)	(c)
0101100	**01011**|00	0101101
1100101	11001|**01**	1100100

Figure 3 An example of a one-point crossover for 7-bit strings: (a) two strings selected for crossover, (b) a crossover site is selected at random, (c) the two strings are swapped after the 5th bit

Finally, after crossover, the *mutation* operator is applied with uniform probability p_m. The operation is a stochastic bit-wise complementation and serves the important, but secondary role of ensuring that the entire representation space Ω remains accessible. Mutation may be applied to offspring produced by crossover or, as an independent operator, at random to any individual in the intermediate population. It operates independently on each individual by probabilistically perturbing each bit string. The event that the ith bit of the kth individual is flipped from 0 to 1 or from 1 to 0 is stochastically independent and occurs with probibility $p_m \in [0, 1]$. As an example, assume $p_m = 0.1$, and the string 0101101 is to undergo mutation. The easiest way to determine which bits, if any, to flip is to select a uniform random number $r \in [0, 1]$ for each bit in the string. If $r \leq p_m = 0.1$, then the bit is flipped, otherwise not. Suppose that the random numbers (0.30, 0.91, 0.05, 0.54, 0.48, 0.89, 0.17) were generated for the string in question, then the third bit has to be flipped and the resulting string is 01**1**1101. After mutation, the candidate strings are copied into the new population $S(t + 1)$ of strings, and the whole process is repeated by decoding each individual into a form appropriate for evaluation, calculating its fitness, using a roulette wheel method of selection, and applying the operators of crossover and mutation.

It is important to note that as the average evaluation of the strings in the population increases, the variance in fitness decreases in the population. After some generations there may be little difference between the best and worst individual in the population, and the selective pressure based on fitness is correspondingly reduced. This problem can be partially addressed by using some form of fitness scaling (Goldberg 1989). In the simplest case, one can subtract the evaluation of the worst string in the population from the evaluations of all strings in the population. One can now calculate the average string evaluation as well as fitness values using this adjusted evaluation. This will increase the resulting selective pressure.

The general idea to use the GA to select a topology (pattern of connections) for the neural spatial interaction models is illustrated in Figure 4. This approach is inherently computationally demanding because the complete conventional training phase (itself computationally intensive) is required to simply evaluate the fitness of a string (i.e. a neural network topology). But the approach remains reasonably

attractive despite this because of the scarcity of fully automated alternative procedures for optimally selecting the network topology. It is especially useful and effective when one needs to find the optimal topology of a highly complex neural network which cannot be assumed by the trial-and-error or heuristic procedures.

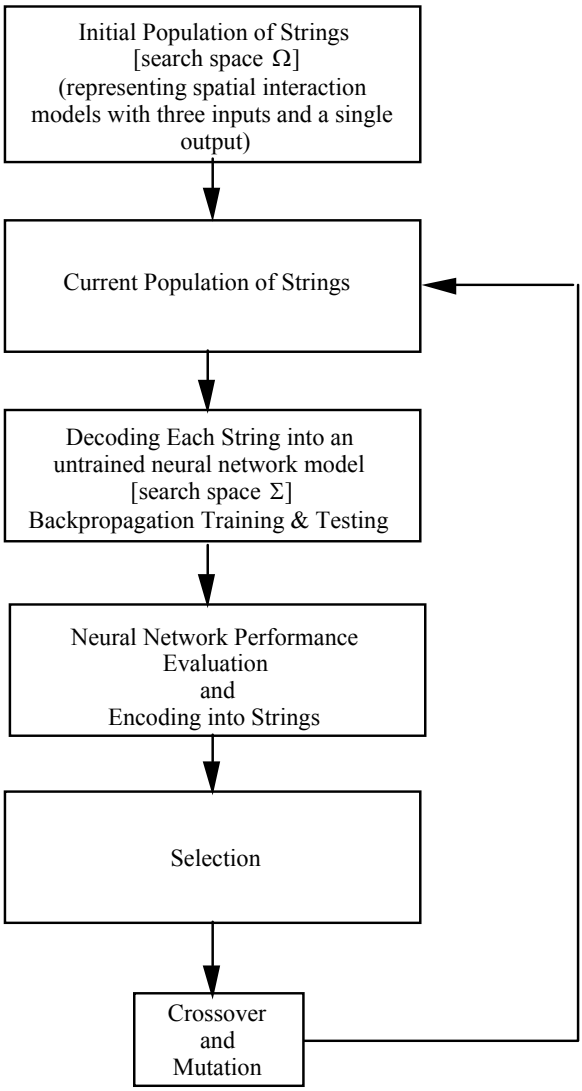


Figure 4 The principle structure of a genetic approach for the neural network topology optimisation problem

While the procedure outlined in Figure 4 is quite straightforward, the problem of combining GA and neural spatial interaction modelling, however, lies in the

definitions of an invertible transformation to encode/decode the original search space Σ into some GA-space Ω, and of the fitness evaluation function. These are the two main components of most genetic algorithms that are problem dependent.

In principle, three major types of direct encoding strategies may be used: node-based, layer-based and pathway-based strategies (opposed to indirect encoding where rules or alikes are encoded that carry information on how the CNN has to be constructed). Node-based strategies do not encode the network weights, but node information such as the number of nodes, connectivity and placement information of the nodes, and the type of transfer functions are encoded. In layer-based encoding schemes the layer size and information about output and input connections are stored inter alia, while in pathway-based encoding, pathways through the network are encoded but not the connections, nodes or layers. It is important to note that the search space Ω (space of strings or representation space) is enormously enlarged if the genetic representation of CNNs distinguishes between networks which differ only by the labelling of hidden units. This problem is known as permutational redundancy associated with the arbitrariness of labels of topologically equivalent hidden nodes, and tends to make genetic navigation very difficult since GAs are sensitive to the potential for redundant representations (Radcliff 1991). Aside from the coding issue, the definition of the fitness function has to be given as part of the problem definition. The fitness function may be defined as a combined measure (= overall performance) which may take into account learning speed, accuracy in terms of out-of-sample (testing) performance and factors such as the size and complexity of the model.

The process of training individual neural network models, measuring their fitness, and applying genetic operators to produce a new population of neural network models is repeated over many generations. Each generation should tend to contain more of the features which were found useful in the previous generation, and an improvement in overall performance can be realised over the previous generation.

The next section serves to discuss the major issues involved in using genetic algorithms for neural network topology optimisation. These include the representation of the strings that specifies both the structure and the estimation procedure, the choice of the underlying space Σ of neural network topologies for exploration, adaptations of the basic genetic operators used to construct meaningful neural spatial interaction structures, and the form of the evaluation function which determines the fitness of a neural network.

4 GENNET: A System for Structure Optimisation and Weight Determination

This section describes the evolutionary algorithm on which the GENNET system is based (Leung et al. 1995). The system essentially consists of two major modules: a Genetic Algorithm Engine used for designing the topology (structure optimisation) and a Network Simulator for gradient-based backpropagation lear-

ning and testing. Both modules are interwoven via a Network Decoder and a Network Fitness Evaluator (see Figure 5). Basically, the Genetic Algorithm Engine encodes neural network topologies as strings and evolves them through genetic operators. The evolved string is then decoded into a neural network by the Network Decoder (generator) and is then fed to the neural network engine for training. Based on the given training patterns, the engine trains the given neural networks by *error backpropagation* of gradient descents, and the resulting networks are then tested with the given testing patterns. Various statistics such as the size of the network, learning speed, in-sample and out-of-sample performance are recorded and passed back to the genetic algorithm engine for fitness evaluation. Networks with high fitness are selected and further processed by various genetic operators. The whole process is repeated until a network with fitness value higher than the specified requirement is found.

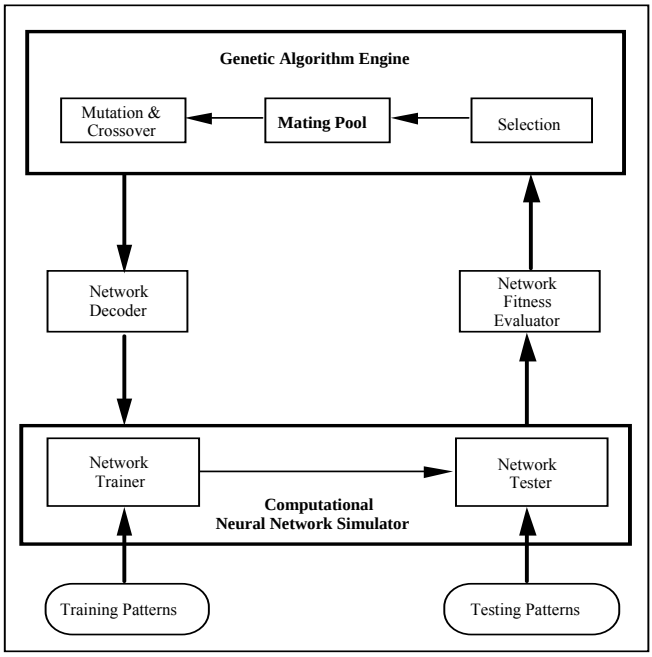


Figure 5 GENNET: A hybrid approach for optimising the structure and the weights of multi-layer computational neural networks

In the sequel, we detail the structure and the mechanism of the GA-based system. The utilisation of genetic algorithms in any specific application entails three related activities: *first*, the definition of an objective function which indicates the fitness of any potential solution, *second*, the definition of the encoding structure, and *third*, the definition of the genetic operators to be used.

4.1 Fitness Function

In GENNET, optimal network design corresponds to the global minimum of a fitness function. The fitness function is defined to depend on three factors. The main factor used for the fitness function is the network's ability to generalise unseen patterns, i.e. the generalisation or out-of-sample performance measured in terms of the average relative variance. For this purpose, we used a validation test set in comparison to the training set. To create a selective pressure that favours smaller networks, a parameter reflecting the network complexity (in terms of the number of hidden layers and the number of their nodes) is also included in the fitness function. Finally, the number of cycles required to train the network (i.e. the speed of convergence) in comparison to the maximum number of cycles utilised so far is taken into consideration, even though this parameter is not directly associated with neural network topology. This leads to the following fitness function

$$Fitness = 1 - r_1\ TestF - r_2\ NCF - r_3\ TrainF \tag{9}$$

where TestF represents the generalisation performance, NCF the network complexity and TrainF the speed of convergence. r_1, r_2 and r_3 are scaling factors between zero and one. They may be interpreted as penalty coefficients and have to be carefully adjusted.

Although the fitness function in (9) is relatively simple, its values depend on several 'hidden' factors which are not directly associated with network topology. For example, evaluating a given network simply on learning results and validation tests has several drawbacks. Using a small number of samples in both phases, for example, speeds up the optimisation process but may result in removing connections too forcefully since the limited number of measurements might not provide enough evidence to justify the importance of some links. On the other hand, using a larger quantity of data to evaluate the CNN may imply that bigger networks could be trained more precisely than smaller ones and, thus, the implicit pruning process would be reluctant to remove links.

4.2 Encoding Scheme

Table 1 illustrates how a string is built. The string representation has several desirable properties. Since all bits are treated uniformly, all genetic operators designed for binary strings can then be applied without modification. The string can be used to encode any initial network architecture. Neural spatial interaction models of class (2) are encoded as a 68-bit string. The following string

1	2	3	4	5	6	
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	12345678
1011110000	**0000000000**	**0001001001**	**1101010101**	**0001000000**	**0000010000**	**00000001**

represents a neural spatial interaction model with a single hidden layer of 30 units (i.e. total number of hidden layers: 1; number of nodes in the hidden layer: 30; connectivity: 1). Its detailed decoding is as follows:

bit	1	2-7	8	9-14	15	16-21	22-24	25-27	28-30	31-40	41-68
binary	1	011110	0	000000	0	0000000	001	001	001	1101010101	00010000000000001000 000000001
decimal	1	30	0	0	0	0	1	1	1	853	16781313

$$\text{upper weight limit} = \tfrac{1}{7} \cdot (\text{range of weight limit})$$
$$\text{lower weight limit} = \tfrac{1}{7} \cdot (\text{range of weight limit})$$
$$\text{momentum limit} = \tfrac{1}{7} \cdot (\text{range of momentum limit})$$
$$\text{probability of connection of each layer} = \frac{852}{2^{10-1}} = \frac{853}{1023} = 0.8338$$
$$\text{sigmoid variable} = 16781313/2^{28-1}$$
$$= 16781313/268435455 \cdot (\text{range of } \lambda)$$

Table 1 Genetic-algorithm encoding of a multi-layer neural network spatial interaction model

Bits	*Meaning*
1	Present flag* of the first hidden layer
2-7	Density parameter: number of nodes in the first hidden layer
8	Present flag* of the second hidden layer
9-14	Density parameter: number of nodes in the second hidden layer
15	Present flag* of the third hidden layer
16-21	Density parameter: number of nodes in the third hidden layer
22-24	Determine upper connection weight limit (has 8 discrete levels)
25-27	Determine lower connection weight limit (has 8 discrete levels)
28-30	Momentum limit (has 8 discrete levels)
31-40	Probability of connection of each layer
41-68	Sigmoid slope (λ) of the logistic function

* The flag is used to indicate whether the hidden layer exists or not; 1: present, and 0: absent

4.3 Operators

The Genetic Algorithm Engine employs the three genetic operators as described in Section 3. For selection, the standard roulette wheel selection operator, with Monte Carlo selection with probabilities based on fitness level is used. Strings with higher fitness values have a higher chance to be selected for reproduction.

For crossover, a random crossover point which is a number between 1 and 67 (string length minus 1) is selected as the position for crossover. For mutation, the standard operator which negates a bit with probability p_m is utilised. Having initialised the evolutionary process with a randomly generated population of strings, the genetic algorithm engine then applies the three genetic operators for reproduction.

All strings generated by the Genetic Algorithm Engine are decoded as neural networks by the *Network Decoder* (generator). At the initialisation of the system, the network decoder is initialised with the following information: maximum number of hidden layers (in the current study: up to one hidden layer), minimum and maximum number of nodes in each layer, lower and upper bounds of the weights. Based on the information, the network decoder converts the string into a multilayer neural network. Each weight of the connections is initialised with a random value between the lower and upper bounds of the weights. The decoded network is fed to the network trainer for backpropagation training.

The *Network Trainer* is responsible for the training of the neural network by backpropagating gradient descent errors and using the training set, provided until any one of the following conditions is fulfilled: number of maximum training cycles is reached, mean squared error (i.e. the error function to be minimised during training) is smaller than the desired value, or the mean error improvement with a cycle period is converged. When the training is completed, the Network Trainer outputs a fitness value: Training Fitness (TrainF), a value within (0, 1) which corresponds to how well the neural network is trained (measured in terms of the error function in (3)). We assume that the smaller the mean squared error, the greater is the TrainF.

The *Network Tester* is responsible for the testing of the trained network using the testing validation set provided. When the testing is finished, the out-of-sample performance or Testing Fitness (TestF), which is a value within (0, 1), is obtained. TestF, which is monitored by the Network Tester, is measured in terms of the average relative performance (ARV) defined by Fischer and Gopal (1994) as

$$\text{ARV} = \frac{\sum_l \| \boldsymbol{y}^l - \Phi^L(\boldsymbol{x}^l, \boldsymbol{w}) \|^2}{\sum_l \| \boldsymbol{y}^l - \bar{\boldsymbol{y}} \|^2} \tag{10}$$

where $(\boldsymbol{x}^l, \boldsymbol{y}^l)$ denotes the lth testing pattern, $\bar{\boldsymbol{y}}$ the average of the target values $\boldsymbol{y}^l$ in the testing data set, and $\Phi^L(\boldsymbol{x}^l, \boldsymbol{w})$ the actual model output. This performance measure provides a normalised mean squared error metric for comparing the generalisation performance of different CNN models.

The tested network is then evaluated by the *Network Fitness Evaluator* that is responsible for the evaluation of the overall fitness of the neural network. Network Fitness (NF) is based on the Network Complexity Fitness (NCF) (defined as size of the network/maximum possible size), Training Fitness (TrainF), and Testing Fitness (TestF), and is defined as

$$\text{NF} = r_{\text{TestF}} \; \mathit{TestF} \; + \; r_{\text{NCF}} \; \mathit{NCF} \; + \; r_{\text{TrainF}} \; \mathit{TrainF} \tag{11}$$

where r_{TestF}, r_{NCF}, and r_{TrainF} are constants which reflect the degrees of significance of NCF, TrainF, and TestF respectively. Adjusting the value of, r_{TestF}, r_{NCF}, and r_{TrainF} is crucial because the system uses these values as a basis to evolve neural networks of various topologies.

The trained networks are then fed to the GA engine for evolution, especially on the adjustment of the various weights in NF, momentum, probability of connection, and the parameter λ of the logistic function. It should be noted that the relationship between fitness in (9) and NF in (11) of strings (i.e. between the GA-search space Ω and the search space Σ of CNN topologies) is given by

$$fitness = 1 - NF. \tag{12}$$

5 Experiments and Performance Tests Using Interregional Telecommunication Traffic Data

Fischer and Gopal (1994) have demonstrated the feasibility of the class of neural spatial interaction models Φ^L to model interregional telecommunication traffic in Austria with noisy real world data of limited record length. Even though the model identified in this study well outperformed current best practice in form of the classical regression approach of the gravity type, the model selection approach utilised, due to its trial-and-error heuristics, might have considered only restricted subsets of the whole space of CNN topologies. Thus, it is obvious to test and evaluate the suggested approach in this spatial interaction context. To facilitate comparison with the previous work, we considered the same class of neural interaction models with three inputs, a single hidden layer, and a single output unit that represents the intensity of telecommunication flows from one origin region to a destination region. The input units represent the three independent variables of the classical gravity model (i.e. the potential pool of telecommunication activities in the origin region, the potential draw of telecommunication activities in the destination region, and a factor representing the inhibiting effect of geographic separation from the origin to the destination region). The problem is to identify a model specification with an appropriate complexity in terms of the hidden units to show good generalisation (i.e. out-of-sample) performance.

5.1 The Data

From three Austrian data sources – a (32, 32)-interregional telecommunication flow matrix, a (32, 32)-distance matrix, and gross regional products for the 32 telecommunication regions – a set of 992 4-tuples (x_1, x_2, x_3, y) was constructed, where the first three components represent the input vector $\boldsymbol{x} := (x_1, x_2, x_3)$ and the last component the target output of the CNN model, i.e. the telecommunica-

tion intensity from one region of origin to another region of destination. Input and target output signals were preprocessed to logarithmically transformed data scaled into [0, 1]. The telecommunication data stem from network measurements of carried telecommunication traffic in Austria 1991, in terms of erlang, which is defined as the number of phone calls (including facsimile transmissions) multiplied by the average length of the call (transfer) divided by the duration of measurement (for more details see Fischer and Gopal 1994). This data set was randomly divided into two separate subsets: about two thirds of the data were used for parameter estimation only, and one third as validation test set. There was no overlapping of the two sets of data.

5.2 Results

A first set of tests was run in order to identify an optimal parameter setting. The genetic algorithm has four parameters and there is no way to a priori identify useful combinations of values. The parameters are: P, the size of the population; p_c, crossover probability; p_m, mutation probability; and λ, slope of the logistic transfer function (see Equation (2)). In order to define good settings, extensive computational tests with different combinations of values have been performed. All tests were made by letting the algorithm run five times at each setting for 500 function evaluations. The values used for each parameter were: $P \in \{20, 40, 100, 200\}$; $p_c = \{0.5, 0.6, 0.7, 0.8, 0.9\}$; $p_m = \{0.001, 0.01, 0.1, 0.2\}$; $\lambda = \{0.1, 0.5, 1, 2, 10\}$. The best settings obtained are summarised in Table 2.

Some considerations are in order. *First*, larger populations provide better results, because a large population is more likely to contain representatives from a larger number of hyperplanes. Thus, the GAs can perform a more informed search. But the computational costs of evolving large populations does not seem to be rewarded by comparable improvement of the solutions obtained by, for $P > 40$. *Second*, there is evidence that, within the representation used, the search process benefits from the recombination. Crossover probability equals to the commonly adopted values suggested, for example, by De Jong (1975), i.e. $p_c = 0.6$. If the crossover probabilities are too high, high performance structures are discarded faster than selection can produce improvements. If the probability is too low, the search may stagnate due to the lower exploration rate. *Third*, mutation is a secondary search operator that increases the variability of the population. The mutation probability is small $(p_m = 0.001)$, suggesting a larger disruptive impact of the mutations it controls. *Fourth*, the sigmoid slope is a parameter that dramatically affects the results. The best slope value, i.e. $\lambda = 2$, corresponds to a steeper sigmoid. The experiments also suggest that in smaller populations such as $P = 40$ structures, good performance is associated with either a higher crossover probability combined with a low mutation probability or a lower crossover combined with a higher mutation probability.

Table 2 Best parameter settings

Parameter	*Values*
P	40
p_c	0.6
p_m	0.001
λ	2
η (learning rate)	0.7
μ (momentum)	0.9

Table 3 reports the generalisation (out-of-sample) performance (in terms of ARV) along with the network complexity (J = 26 and J = 36) of the CNNs evolved over the five trials. Again some considerations are worth making. *First*, compared with the results obtained in Fischer and Gopal (1994) the solutions show a higher variance over the five runs. Sometimes, it moves steadily (though slowly) towards the optimum, sometimes it never greatly modifies the initial fitness, which is close to the half of the optimum. *Second*, the average performance in terms of ARV obtained over the five runs utilising the validation test patterns is 0.3948 in the case of the less complex network model (J = 26) and 0.3879 in the case of the more complex model (J = 30). *Third*, for comparative purposes it is worthwhile to mention that the cross-validation approach utilised in Fischer and Gopal (1994) did result in a model with J = 30 hidden units and an ARV value of 0.4065 on the test rather than on the validation set averaged over five trials differing in initial conditions as well as in the random sequence of the input signals. But it should be noted that the ARV values are not comparable in a strict sense, because Fischer and Gopal (1994) utilise an additional set of test data, independent from both the training and the validation test sets. The test data set is reserved for the training process, the validation test set for optimising model complexity (i.e. model choice) and the test set for the evaluation.

Table 3 Performance of the GA solutions (measured in terms of ARV) using the validation test set

	Neural spatial interaction model with	
	J=26	*J=30*
	Average relative variances	*Average relative variances*
Run 1	0.588735	0.554846
Run 2	0.306277	0.295533
Run 3	0.294214	0.275657
Run 4	0.417048	0.421113
Run 5	0.367715	0.392486
Average	0.394798	0.387927
Standard deviation	0.119157	0.111943

The best network model should be a compromise between complexity and performance. In the suggested GA model choice approach, the balance between complexity and performance can be regulated at the level of the fitness function. This is an attractive feature. But care has to be exercised when setting the *r*-values since the network performance depends also on other 'hidden' aspects of the optimisation problem, such as the number of training and validation points in addition to the overall training strategy.

6 Conclusions and Outlook

This paper presents an evolutionary computational approach, called GENNET, suitable for maximising complex functions, such as those that assign a fitness to a multi-layer computational neural network on the basis of its topology, its training performance and convergence speed. Search effectiveness is achieved through a direct encoding procedure that allows CNN topologies to be represented by 68-bits strings, and an algorithm that automatically identifies minimal search space containing reachable solutions at evolution time. The genetic algorithm permits search over the whole search space of CNN topologies, thus, providing the possibility of escaping from local minima. Local search is performed by applying backpropagation learning to the individuals of the GA-population. Interregional telecommunication traffic data from Austria (Fischer and Gopal 1994) serve as a benchmark for evaluating and illustrating the performance of the hybrid aproach where the objective is to generate a three inputs, single output, and single hidden layer computational neural network with logistic hidden units.

As was shown in the simulation studies, GENNET successfully generates an optimal topology for the neural network spatial interaction model without manual intervention required by other heuristic approaches to the model choice problem. Though the computational costs tend to be higher, it is, in fact cost effective if time spent on human supervision and intervention in the other approaches is taken into account. The benefit will be even more apparent when complex network topologies more complex than that in this study have to be evolved.

We believe that GENNET may be made substantially more efficient, for example, by using floating-point rather than binary representations (see Michalewicz 1992). But there is little doubt that it will remain slower in any serial implementation. The greatest potential for the application of this and other types of evolutionary optimisation to real world problems will come from their implementation on parallel machines, because evolution is an inherently parallel process. While the evolutionary process itself is slow, the CNN it generates is evolved to be computationally efficient in the sense of producing the best approximation using the fewest computational units, provided the system parameters used in the experiments are optimal for the given problem.

Finally, we should mention that there are several other population based algorithms that are either spinoffs of the canonical genetic algorithm, or independently developed. Evolution strategies and evolutionary programming, for example, are

two computational paradigms that use a population based search. Evolutionary programming (see Fogel 1995), in particular, appears to be an attractive alternative to genetic algorithms since it provides the possibility to manipulate CNNs directly and, thus, obviates the need for a dual representation. We leave comparisons between evolutionary programming and genetic algorithms on the problem addressed in this paper for future research.

***Acknowledgements*:** The authors would like to thank K.S. Leung, W. Ng, M.K. Lau and S.K. Cheung for their assistance in implementation; and the Hong Kong Research Grant Council and the Austrian Fonds zur Förderung der wissenschaftlichen Forschung for the support through the earmarked grant CUHK 321/95H and FWF P12681-INF.

References

Bishop C.M. (1995): *Neural networks for pattern recognition*, Clarendon Press, Oxford

Caudele T.P. and Dolan C.P. (1989): Parametric connectivity: Training of constrained networks using genetic algorithms. In: Schaffer J.D. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, Morgan Kaufmann, San Mateo [CA], pp. 370-374.

De Jong K.A. (1975): Analysis of the behavior of a class of genetic adaptive systems, Ph.D. dissertation, University of Michigan

Finnoff W. (1993): Diffusion approximations for the constant learning rate backpropagation algorithm and resistance to local minima. In: Hanson S.J., Cowan J.D. and Giles C.L. (eds.) *Advances in Neural Information Processing Systems V*, Morgan Kaufmann, San Mateo [CA] pp. 459-466

Fischer M.M. (1998): Computational neural networks. A new paradigm for spatial analysis, *Environment and Planning A* 30 (10), 1873-1892

Fischer M.M. and Gopal S. (1994): Artificial neural networks: A new approach to modelling interregional telecommunication flow, *Journal of Regional Science* 34 (4), 503-527

Fischer M.M., Gopal S., Staufer P. and Steinnocher K. (1997): Evaluation of neural pattern classifiers for a remote sensing application, *Geographical Systems* 4 (2), 195-223

Fogel D.B. (1994): An introduction to simulated evolutionary optimization, *IEEE Transactions on Neural Networks* 5 (1), 3-14

Fogel D.B. (1995): *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway [NJ]

Goldberg D. (1989): *Genetic Algorithms*, Addison-Wesley, Reading [MA]

Gopal S. and Fischer M.M. (1996): Learning in single hidden layer feedforward network models, *Geographical Analysis* 28 (1), 38-55

Grefenstette J.J. (1986): Optimisation of control parameters for genetic algorithms, *IEEE Transactions on Systems, Man and Cybernetics* SMC 16, 122-128

Hassoun M.H. (1995): *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge [MA] and London [England]

Hinton G.E. (1990): Connectionist learning procedures In: Kodratoff Y. and Michalski R. (eds.) *Machine Learning III*, Morgan Kaufmann, San Mateo [CA], pp. 555-610

Holland J.H. (1975): *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor (Michigan)

Hornik K.M., Stinchcombe M. and White H. (1989): Multi-layer feedforward networks are universal approximators, *Neural Networks* 2, 359-366

Koza J.R. (1993): *Genetic Programming*, MIT Press, Cambridge [MA] and London [England]

Leung Y. (1997): Feedforward neural network models for spatial pattern classification. In: Fischer M.M. and Getis A. (eds.) *Recent Developments in Spatial Analysis*: *Spatial Statistics, Behavioural Modelling, and Computational Intelligence*, Springer, Berlin, Heidelberg, New York, pp. 336-359

Leung Y. (1994): Inference with spatial knowledge: An artificial neural network approach, *Geographical Systems* 1 (2), 103-121

Leung Y., Leung K.S., Ng W. and Lau M.I. (1995): Evolving multi-layer feedforward neural networks by genetic algorithms (unpublished paper)

Maniezzo V. (1994): Genetic evolution of the topology and weight distribution of neural networks, *IEEE Transactions on Neural Networks* 5 (1), 39-53

Michalewicz Z. (1992): *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, Heidelberg, New York

Miller G.F., Todd P.M. and Hedge S.U. (1989): Designing neural networks using genetic algorithms. In: Schaffer J.D. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, Morgan Kaufmann, San Mateo [CA], pp. 379-384

Montana D.J. and Davis L. (1989): Training feedforward networks using genetic algorithms. In: Sridhara N.S. (ed.) *Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo [CA], pp. 762-767

Moody J. (1992): Generalisation, weight decay and architecture selection for nonlinear learning systems. In: Moody J., Hanson J. and Lippmann R. (eds.) *Advances in Neural Information Processing Systems IV*, Morgan Kaufmann, San Mateo [CA], pp. 471-479

Nijkamp P., Reggiani A. and Tritapepe T. (1996): Modelling inter-urban flows in Italy: A comparison between neural network approach and logit analysis, *Transportation Research C* 4 (6), 323-338

Openshaw S. (1998): Neural network, genetic, and fuzzy logic models of spatial interaction, *Environment and Planning A* (30), 1857-1872

Openshaw S. (1993): Modelling spatial interaction using neural net. In: Fischer M.M. and Nijkamp P. (eds.) *Geographical Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 147-164

Openshaw S. (1988): Building an automated modelling system to explore a universe of spatial interaction models, *Geographical Analysis* 20 (1), 31-46

Openshaw S. and Openshaw C. (1997): *Artificial Intelligence in Geography*, John Wiley, Chichester [UK], New York

Qi X. and Palmieri F. (1994): Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space, Part I: Basic properties of selection and mutation, *IEEE Transactions on Neural Networks* 5 (1), 102-119

Radcliffe N.J. (1991): Genetic set recombination and its application to neural network topology optimisation, EPCC Technical Report EPCC-TR-91-21, Edinburgh Parallel Computing Centre, University of Edinburgh

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

Schaffer J.D., Caruana R.A., Eshelman L.J. and Das R. (1989): A study of control parameters affecting online performance of genetic algorithms for function optimisation.

In: Schaffer J.D. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, Morgan Kaufmann, San Mateo [CA], pp. 51-60

Spears M.M. and De Jong K.A. (1991): An analysis of multi-point crossover. In: Rawlins G.J.E. (ed.) *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo [CA], pp. 301-315

Syswerda G. (1989): Uniform crossover in genetic algorithms. In: Schaffer J.D. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo [CA], pp. 2-8

White H. (1989): Learning in artificial neural networks: A statistical perspective, *Neural Computation* 1, 425-464

Whitley D. and Hanson T. (1989): Optimising neural networks using faster, more accurate genetic search. In: Schaffer J.D. (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo [CA], pp. 391-396

Yao X. (1993): A review of evolutionary artificial neural networks, *International Journal of Intelligent Systems* 8 (4), 539-567

Part III

GeoComputation in Remote Sensing Environments

9 Evaluation of Neural Pattern Classifiers for a Remote Sensing Application

This paper evaluates the classification accuracy of three neural network classifiers on a satellite image-based pattern classification problem. The multispectral pattern classification task is to assign pixels to one of eight prespecified urban land use categories on a pixel-by-pixel basis. The neural network classifiers used include two types of the Multi-Layer-Perceptron (MLP) and the Radial Basis Function Network. A normal (conventional) classifier is used as a benchmark to evaluate the performance of neural network classifiers. The satellite image consists of 2,460 pixels selected from a section (270 x 360) of a Landsat-5 TM scene from the city of Vienna and its northern surroundings. In addition to evaluation of classification accuracy, the neural classifiers are analysed for generalisation capability and stability of results. The best result in terms of classification accuracy is provided by the MLP-1 classifier with weight elimination. It has a small number of parameters and requires no problem-specific system of initial weight values. Its in-sample classification error is 7.87% and its out-of-sample classification error is 10.24% for the problem at hand. Four classes of simulations have been undertaken to illustrate the properties of the classifier in general and the stability of the result with respect to control parameters, such as the gradient descent control term, initial parameter conditions, and different training and testing sets.

1 Introduction

Satellite remote sensing, developed from satellite technology and image processing, has been a popular focus of pattern recognition research since at least the 1970s. Most satellite sensors used for land applications are of the imaging type and record data in a variety of spectral channels and at a variety of ground resolutions. The current trend is for sensors to operate at higher spatial resolutions and for providing more spectral channels to optimise the information content and the usability of the acquired data for monitoring, mapping and inventory applications. At the end of this decade, the image data obtained from sensors on the currently operational satellites will be augmented by new instruments with many more spectral bands on board of polar orbiting satellites forming part of the Earth Observing System (Wilkinson et al. 1994).

As the complexity of satellite data grows, so too does the need for new tools to analyse them in general. Since the mid 1980s, neural network (NN) techniques have raised the possibility of realising fast, adaptive systems for multispectral

satellite data classification. In spite of the increasing number of NN-applications in spectral pattern recognition (see, for example, Key et al. 1989, Benediktsson et al. 1990, Hepner et al. 1990 Lee et al. 1990, Bischof et al. 1992, Heerman and Khazenie 1992, Civco 1993, Dreyer 1993, Salu and Tilton 1993, Wilkinson et al. 1994) very little has been done on evaluating different classifiers. Given that pattern classification is a mature area and that several NN approaches have emerged in the last few years, the time seems to be ripe for an evaluation of different neural classifiers by empirically observing their performance on a larger data set. Such a study should not only involve at least a moderately large data set, but should also be unbiased. All the classifiers should be given the same feature sets in training and testing.

This paper addresses the above mentioned issue in evaluating the classification accuracy of three neural network classifiers. The classifiers include two types of the Multi-Layer Perceptron (MLP) and a Radial Basis Function Network (RBF). The widely used normal classifier based on parametric density estimation by maximum likelihood, NML, serves as benchmark. The classifiers were trained and tested for classification (eight a priori given classes) of multispectral images on a pixel-by-pixel basis. The data for this study was selected from a section (270 x 360 pixels) of a Landsat-5 Thematic Mapper scene (TM Quarter Scene 190-026/4; location of the center: 16° 23' E, 48° 14' N; observation date: June 5, 1985).

In Section 2 of this paper, we will describe the structure of the various pattern classifiers. Then we will briefly characterise the remote sensing classification problem and outline the experimental set-up in Section 3, i.e., the essential organisation of inputs and outputs, the network set-ups of the neural classifiers, a technique for addressing the problem of overfitting, criteria for evaluating the estimation (in-sample) and generalisation (out-of-sample) ability of the different neural classifiers and the simulation set up. Four classes of simulations serve to analyse the stability of the classification results with respect to training time (50,000 epochs), the gradient descent control term (constant and variable learning schemes), the initial parameter conditions, and different training and testing sets. The results of the experiments are presented in Section 4. Finally, in Section 5 we give some concluding remarks.

2 The Pattern Classifiers

Each of our experimental classifiers consists of a set of components as shown in Figure 1. The ovals represent input and output data, the rectangles processing components, and the arrows the flow of data. The components do not necessarily correspond to separate devices. They only represent a separation of the processing into conceptual units so that the overall structure may be discerned. The inputs may – as in the current context – come from Landsat-5 Thematic Mapper (TM) bands.

Each classifier provides a set of discriminant functions D_c ($1 \leq c \leq C$, C number of a priori given classes). There is one discriminant function D_c for each class c. Each one provides a single floating-point-number which tends to have a

large number if the input pixel (i.e. feature vector $\boldsymbol{x}$ of the pixel, $\boldsymbol{x} \in \mathfrak{R}^n$) is of the class corresponding to that particular discriminant function. The C-tuple of values produced by the set of discriminant functions is sent to the 'Maximum Finder'. The 'Maximum Finder' identifies which one of the discriminant values $D_c(\boldsymbol{x})$ is highest, and assigns its class as the hypothesised class of the pixel, i.e. uses the following decision rule

$$\text{Assign } \boldsymbol{x} \text{ to class } c \text{ if } D_c(\boldsymbol{x}) > D_k(\boldsymbol{x}) \qquad \text{for } k = 1, ..., C \text{ and } k \neq c \tag{1}$$

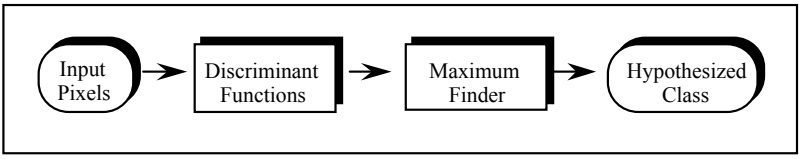


Figure 1 Components of the pixel-by-pixel classification system

Three experimental neural classifiers are considered here: multi-layer perceptron (MLP) classifiers of two types, MLP-1 and MLP-2, and one radial basis function (RBF) classifier. The normal classifier NML serves as statistical benchmark. The following terminology will be used in the descriptions of the discriminant functions below:

n dimensionality of feature space (n representing the number of spectral bands used, n = 6 in our application context),

$\mathfrak{R}^n$ the set of all n-tuples of real numbers (feature space),

$\boldsymbol{x}$ feature vector of a pixel $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathfrak{R}^n$,

C number of a priori given classes ($1 \leq c \leq C$).

2.1 The Normal Classifier

This classifier (termed NML) which is most commonly used for classifying remote sensing data serves as benchmark for evaluating the neural classifiers in this paper. NML is based on parametric density estimation by maximum likelihood (ML). It presupposes a multivariate normal distribution for each class c of pixels. In this context, it may be worthwhile to mention first factors pertaining to any parametric classifier.

Let $L(c|k)$ denote the loss (classification error) incurred assigning a pixel to class c rather than to class k. Let us define a particular loss function in terms of the Kronecker symbol δ_{ck}

$$L(c \mid k) = 1 - \delta_{ck} = \begin{cases} 0 & c = k \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

This loss function implies that correct classifications yield no losses, while incorrect classifications produce equal loss values of 1. In this case the optimal or Bayesian classifier is that one which assigns each input $\boldsymbol{x}$ ('feature vector' of a pixel), to that class c for which the a posteriori probability $p(c \mid \boldsymbol{x})$ is highest, i.e.

$$p(c \mid \boldsymbol{x}) \geq p(k \mid \boldsymbol{x}) \qquad k = 1, ..., C. \tag{3}$$

According to Bayes rule

$$p(c \mid \boldsymbol{x}) = \frac{p(c)\ p(\boldsymbol{x} \mid c)}{p(\boldsymbol{x})} \tag{4}$$

where $p(c)$ denotes the a priori probability of class c and $p(\boldsymbol{x})$ the mixture density $\int p(\boldsymbol{x})\, d\boldsymbol{x}$ with $\boldsymbol{x}$ belonging to the training set $S \subseteq \mathfrak{R}^n$. For a pattern classification problem in which the a priori probabilities are the same, $p(c)$ can be ignored. For the normal classifier NML each class c is assumed to have a conditional density function

$$p(\boldsymbol{x} \mid c) = (2\pi)^{-\frac{n}{2}} \mid \boldsymbol{\Sigma}_c \mid^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_c)^{\mathrm{T}} \boldsymbol{\Sigma}_c^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_c)\right\} \qquad c = 1, ..., C \tag{5}$$

with $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ being the mean and associated covariance matrix for class c. The first term on the right-hand side of (5) is constant and may be discarded for classification. By replacing the mean vectors $\boldsymbol{\mu}_c$ and the covariance matrices $\boldsymbol{\Sigma}_c$ with their sample estimates, $\boldsymbol{m}_c$ and $\boldsymbol{S}_c$, squaring and taking logarithms the set of NML-discriminant functions is given by

$$D_c(\boldsymbol{x}) = 2 \log \hat{p}(c) - \log \mid \boldsymbol{S}_c \mid - (\boldsymbol{x} - \boldsymbol{m}_c)^{\mathrm{T}}\ \boldsymbol{S}_c^{-1}\ (\boldsymbol{x} - \boldsymbol{m}_c) \tag{6}$$

where $\hat{p}(c)$ denotes the estimate of $p(c)$.

2.2 The Multi-Layer Perceptron Classifiers

Multi-layer perceptrons are feedforward networks with one or more layers of nodes between the input and the output nodes. These additional layers contain hidden (intermediate) nodes or units. We have used MLPs with three layers (counting the inputs as a layer), as outlined in Figure 2.

Let $N^{(k)}$ denote the number of units in the kth layer (k = 0, 1, 2). The number of inputs, $N^{(0)}$ [= n] and the number of outputs, $N^{(2)}$ [= C] are determined by the

application at hand, and in our study are six for the input layer (one for each spectral channel TM1, TM2, TM3, TM4, TM5 and TM7) and eight for the output layer (representing the eight a priori categories of the pixels). The parameter with respect to the network architecture outlined in Figure 2 is the number $N^{(1)}$ of nonlinear hidden units that are fully connected to the input units and with the output units. Output and hidden units have adjustable biases (left out of consideration in Figure 2). The weight $\omega_{ji}^{(l)}$ connects the ith node of the (l-1)-th layer to the jth node of the lth layer (l = 1, 2; $1 \le I \le N^{(l-1)}$, $1 \le j \le N^{(l)}$). The weights can be positive, negative or zero.

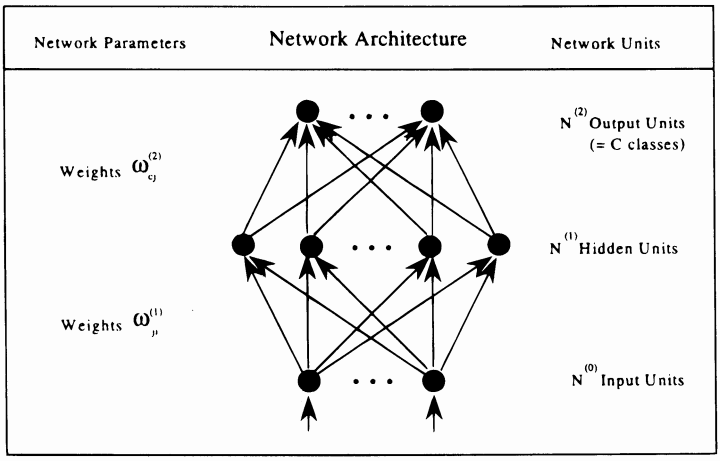


Figure 2 Architecture of a $N^{(0)}{:}N^{(1)}{:}N^{(2)}$ perceptron

Let us define $b_c^{(l)}$ the bias term of the ith node of the lth layer (l = 1, 2), and $\psi(\boldsymbol{x})$ the nonlinear hidden unit activation function, then the set of discriminant functions are of the form:

$$D_c(\boldsymbol{x}) = \frac{\exp\left\{b_c^{(2)} + \sum_{j=1}^{N^{(1)}} \omega_{cj}^{(2)} \ \psi(b_j^{(1)} + \sum_{i=1}^{N^{(0)}} \omega_{ji}^{(1)} \ x_i)\right\}}{\sum_{i=1}^{N^{(2)}} \exp\left\{b_1^{(2)} + \sum_{j=1}^{N^{(1)}} \omega_{1j}^{(2)} \ \psi(b_j^{(1)} + \sum_{i=1}^{N^{(0)}} \omega_{jk}^{(1)} \ x_k)\right\}} \qquad c = 1, \ldots, C \qquad (7)$$

It is worthwhile to note that classifiers of type (7) use a softmax output unit activation function (see Bridle 1989). This activation function is a composition of two operators: an exponential mapping, followed by a normalisation to ensure that the output activations are non-negative and sum to one. The specification of the activation function ψ is a critical issue in successful application development of a MLP classifier. We have experimented with two types of sigmoid functions, the most widely used nonlinear activation functions: asymmetric and symmetric sigmoid

functions. We use logistic activation for defining MLP-1 and hyperbolical tangent (tanh) activations for MLP-2.

The activation S_h of a logistic (sigmoid) hidden unit is given by

$$S_h = \left\{ 1 + \exp\left[- a \, (b_h^{(1)} + \sum_{i=1}^{N^{(0)}} \omega_{hi}^{(1)} \; x_i) \right] \right\}^{-1} \tag{8}$$

which performs a smooth mapping $(-\infty, +\infty) \rightarrow (0, 1)$. The slope '$a$' can be absorbed into weights and biases without loss of generality and is set to one.

The activation T_h of a tanh hidden unit is given by

$$T_h = \tan\left[a \, (b_h^{(1)} + \sum_{i=1}^{N^{(0)}} \omega_{hi}^{(1)} \; x_i) \right] \tag{9}$$

performing a smooth mapping $(-\infty, +\infty) \rightarrow (-1, +1)$. We here also set $a = 1$.

For the training of the weights of MLP networks, a reasonable procedure is the use of an optimisation algorithm to minimise the mean-square-error (least squares error function) over the training set between the discriminant values actually produced and the target discriminant values that consist of the appropriate strings of 1s and 0s as defined by the actual classes of the training pixels. For example, if a training vector is associated to class 1, then its target vector of discriminant values is set to (1, 0, ..., 0).

Networks of the MLP type are usually trained using the error backpropagation algorithm (see Rumelhart et al. 1986). Error backpropagation is an iterative gradient descent algorithm designed to minimise the least squares error between the actual and target discrimination values. This is achieved by repeatedly changing the weights of the first and second parameter layer according to the gradient of the error function. The updating rule is given by

$$\omega_{rs}^{(k)}(t+1) = \omega_{rs}^{(k)}(t) + \eta \, \frac{\partial E}{\partial \omega_{rs}^{(k)}} \qquad k = 1, 2 \tag{10}$$

where E denotes the least squares error function to be minimised over the set of training examples, and η the learning rate, i.e. the fraction by which the global error is minimised during each pass. The bias value b_h is also learned in the same way. In the limit, as η tends to zero and the number of iterations tends to infinity, this learning procedure is guaranteed to find the set of weights which gives the least squares error (see White 1989).

2.3 The Radial Basis Function Classifier

In the MLP classifiers, the net input to the hidden units is a linear combination of the inputs. In a Radial Basis Function (RBF) network the hidden units compute radial basis functions of the inputs. The net input to the hidden layer is the distance from the input to the weight vector. The weight vectors are also called centres. The distance is usually computed in the euclidean metric. There is generally a band width σ associated with each hidden unit. The activation function of the hidden units can be any of a variety of functions on the non-negative real numbers with a maximum at zero, approaching zero at infinity, such as the Gaussian function.

We have experimented with a RBF classifier which uses softmax output units and Gaussian functions in the hidden layer. The following notation is necessary to describe the classifier. Let

$$\boldsymbol{c}^{(k)} = (\boldsymbol{c}_1^{(k)}, ..., \boldsymbol{c}_n^{(k)})^{\mathrm{T}} \in \mathfrak{R}^n \qquad k = 1, ..., N^{(1)}$$

denote the centre vector of the *k*th hidden unit and

$$\sigma^{(k)} = (\sigma_1^{(k)}, ..., \sigma_n^{(k)})^{\mathrm{T}} \in \mathfrak{R}^n \qquad k = 1, ..., N^{(1)}$$

its width vector, while $b_k^{(1)}$ and $\omega_{lk}^{(2)}$ with $1 \le l \le N^{(2)} =: C$ and $1 \le k \le N^{(1)}$ and are the bias term to the *k*th node of the *l*th layer and the weight connecting the *l*th output node to the *k*th hidden node, respectively.

Then the discriminant functions are given by:

$$D_c(\boldsymbol{x}) = \frac{\exp\left\{b_c^{(2)} + \sum_{k=1}^{N^{(1)}} \omega_{ck}^{(2)} \; \Phi_k(\boldsymbol{x})\right\}}{\sum_{l=1}^{N^{(2)}} \exp\left\{b_c^{(2)} + \sum_{k=1}^{N^{(1)}} \omega_{lk}^{(2)} \; \Phi_k(\boldsymbol{x})\right\}} \qquad c = 1, ..., C \tag{11}$$

where each hidden unit *j* computes the following radial basis function:

$$\Phi_k(\boldsymbol{x}) = \exp\left(-\sum_{k=1}^{N^{(1)}}\left(\frac{x_i - c_i^{(k)}}{\sigma_i^{(k)}}\right)\right) = \prod_{i=1}^{N^{(0)}} \exp\left(-\left(\frac{x_i - c_i^{(k)}}{\sigma_i^{(k)}}\right)\right) \qquad k = 1, ..., N^{(1)} \tag{12}$$

The centres $\boldsymbol{c}^{(k)}$, widths $\sigma^{(k)}$, output bias nodes $b_c^{(2)}$ and output node weights $\omega_{lk}^{(2)}$ may be considered as trainable weights of the RBF network. They are trained initially using the cluster means (obtained by means of the *k*-means algorithm) as the centre vectors $\boldsymbol{c}^{(k)}$. The width vectors $\sigma^{(k)}$ are set to a single tunable positive value. Note that no target discriminant values are used to determine $\boldsymbol{c}^{(k)}$ and $\sigma^{(k)}$

while training of the output weights and bias proceeds by optimisation identical to that described for the MLP classifiers.

The crucial difference between the RBF and the two MLP classifiers lies in the treatment of the inputs. For the RBF classifier, as can be seen from (12), the inputs factor completely. Unless all inputs $x_i (1 \leq i \leq n)$ are reasonably close to their centres $c_i^{(k)}$, the activation of hidden unit k is close to zero. A RBF unit is shut off by a single large distance between its centre and the input in any one of the dimensions. In contrast, in the case of the MLP classifiers, a large contribution by one weighted output in the sum of (7) or (8) can often be compensated for by the contribution of other weighted inputs of the opposite sign. This difference between MLP and RBF classifiers increases with the dimensionality of the feature space.

3 Experimental Set up

3.1 The Data and Data Representation

The data used for training and testing the classification accuracy of the classifiers was selected from a section (270 x 360 pixels) of a Landsat 5 Thematic Mapper (TM) scene. The area covered by this imagery is 8.1 x 10.8 km^2 and includes the city of Vienna and its northern surroundings. The spectral resolution of each of the six TM bands (TM1, TM2, TM3, TM4, TM5, TM7) which were used in this study was eight bits of 256 possible digital numbers. Each pixel represents a ground area of 30 x 30 m^2 The purpose of the multispectral image classification task was to distinguish between eight land cover categories as outlined in Table 1.

One of the authors, an expert photo interpreter with extensive field experience of the area covered by the image, used ancillary information from maps and orthophotos (from the same time period) in order to select suitable training sites for each class. One training site was selected for each of the eight categories of land-cover (single training site case). This approach resulted in a database consisting of 2,460 pixels (about 2.5 percent of all the pixels in the scene) that are described by six-dimensional feature vectors and their class membership (target values). The set was divided into a training set (two thirds of the training site pixels) and a testing set by stratified random sampling, stratified in terms of the eight categories. Thus each training/test run consists of 1.640 training/820 testing vectors. This moderately large size for each training run makes the classification problem non-trivial at the one hand, but still allows for extensive tests on in-sample and out-of-sample performance of the classifiers.

Data preprocessing (i.e. filtering or transforming the raw input data) plays an integral part in any classification system. Good preprocessing techniques reduce the effect of poor quality (noisy) data and this usually results in improved classification performance. In this study, the classifiers implemented in the experiments use grey coded data. The grey scale values in each spectral band were linearly compressed in the (0.1, 0.9) range to generate the input signals.

Table 1 Categories used for classification and number of training/testing pixels

Category number	*Description of the category*	*Pixels*	
		Training	*Testing*
*C*1	Mixed grass and arable farmland	167	83
*C*2	Vineyards and areas with low vegetation cover	285	142
*C*3	Asphalt and concrete surfaces	128	64
*C*4	Woodland and public gardens with trees	402	200
*C*5	Low density residential and industrial areas (suburban)	102	52
*C*6	Densely built up residential areas (urban)	296	148
*C*7	Water courses	153	77
*C*8	Stagnant water bodies	107	54
Total number of pixels for training and testing		1,640	820

3.2 Network Set Up of the Neural Classifiers and the Overfitting Problem

The architecture of a neural classifier is defined by the arrangement of its units, i.e. the set of all weighted connections between units (see Figure 2). This arrangement (i.e. the topology) of the network of a classifier is very important in determining its generalisation ability. Generalisation refers to the ability of a classifier to recognise patterns outside the training set. An important issue for good generalisation is the choice of the optimal network size. This means finding the optimal number of hidden units, since inputs and outputs are defined by the problem at hand. There are some rules of thumb which often fail drastically since they ignore both the complexity of the task at hand and the redundancy in the training data (Weigend 1993). The optimal size of the hidden layer is usually not known in advance.

The number of hidden units when the minimum is arrived may be viewed as a kind of measure of the degree of freedom of the network (Gershenfield and Weigend 1993). If the hidden layer is chosen to be too small, it will not be flexible enough to discriminate the patterns well, even in the training set. If it is chosen too large, the excess freedom will allow the classifier to fit not only the signals, but also the noise. Both, too small and too large hidden layers thus lead to a poor generalisation capability in the presence of noise (Weigend et al. 1991).

This issue of overfitting or in other words the problem of estimating the network size has been widely neglected in remote sensing applications, up to now. Recently, several techniques have been proposed to get around this problem. To be relieved from the uncertainty of a specific choice of a validation set of the cross-validation approach (see Fischer and Gopal 1994) we have chosen in this study another approach, a network pruning or weight-elimination technique to overcome the problem of overfitting. This technique starts with an oversized

network and attempts to minimise the complexity of the network (in terms of connection weights) and the least squares error function by removing 'redundant' or least sensitive weights (see Weigend et al. 1991).

We deliberately have chosen an oversized, fully connected MLP-1 network with 22 hidden units and a variable learning rate. The 338 weights were updated after each three patterns, presented in random order (stochastic approximation). In the first 17,000 epochs, the procedure eliminated the weights between the eight output units and eight hidden units. Since these eight units did not receive the signals in the backward pass anymore, their weights to the input subsequentially decayed. In this sense, the weight-elimination procedure can be thought of as unit-elimination, removing the least important hidden units. The weights and biases of the pruned MLP with 14 remaining hidden units are given in appendix A. The architecture of the pruned MLP-1 is outlined in Figure 3. The size of the network declined from 338 to 196 free parameters.

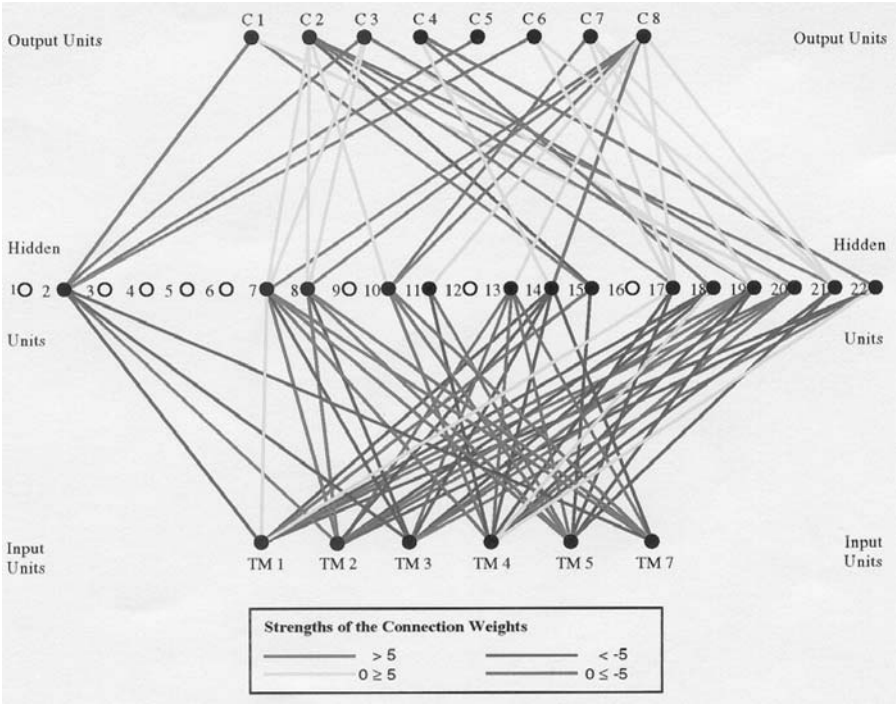


Figure 3 The pruned MLP-1 with 14 degrees of freedom and 196 parameters

In contrast to MLP-classifiers, RBF networks are self-pruning to some degree. Unimportant connections are effectively pruned away by the training process leaving a large width. Each large width effectively deletes one connection from an input to one RBF and reduces the number of active patterns by two.

3.3 Performance Measures

The ultimate performance measure for any classifier is its usefulness to provide accurate classifications. This involves in-sample and especially out-of-sample classification accuracy. Four standard measures will be used to measure various aspects of classification accuracy:

- the *classification error* (also termed confusion) *matrix* ($f_{(lk)}$) with $f_{(lk)}(l, k = 1, ..., C)$ denoting the number of pixels assigned by the classifier to category l and found to be actually in (ground truth) category k,

- the *map user's classification accuracy*, υ_k, for the ground truth category $k = 1, ..., C$

$$\upsilon_k := \frac{f_{kk}}{f_{\bullet k}} := \frac{f_{kk}}{\sum_{i=1}^{C} f_{ik}} \tag{13}$$

- the *map producer's classification accuracy* π_l for the classifier's category $l = 1, ..., C$

$$\pi_l := \frac{f_{ll}}{f_{l\bullet}} := \frac{f_{ll}}{\sum_{j=1}^{C} f_{lj}} \tag{14}$$

- the total *classification accuracy* τ [or the total classification error τ' defined as $\tau' = (100 - \tau)$]

$$\tau := \frac{\sum_{i=1}^{C} f_{ii}}{f_{\bullet\bullet}} := \frac{\sum_{i=1}^{C} f_{ii}}{\sum_{k=1}^{C}\sum_{l=1}^{C} f_{lk}} \tag{15}$$

3.4 Experimental Simulation Set Up

Neural networks are known to produce wide variations in their performance properties. This is to say that small changes in network design, and in control parameters such as the learning rate and the initial parameter conditions might generate large changes in network behaviour. This issue, which is the major focus of our simulation experiments, has been highly neglected in remote sensing applications up to now. In real world applications, it is, however, a central objective to identify

intervals of the control parameters which give robust results, and to demonstrate that these results persist across different training and test sets.

In-sample and out-of-sample performance are the two most important experimentation issues in this study. In-sample performance of a classifier is important because it determines its convergence ability and sets a target of feasible out-of-sample performance which might be achieved by fine-tuning of the control parameters (Refenes et al. 1994). Out-of-sample performance measures the ability of a classifier to recognise patterns outside the training set, i.e. in the testing set strictly set apart from the training set. The performance depends on many factors, such as

- the gradient descent control term,
- initial parameter conditions, and
- training and testing sets.

Consequently, it is important to analyse the stability with respect to such control parameters. Several other important issues are not considered in this study, such as for example the issue of how the convergence speed can be improved. We have not used any acceleration scheme of backpropagation such as momentum. We also do not discuss the dependence of the performance on the size or the number of the training/testing sets.

For our MLP-simulations we used parameter values initialised with uniformly distributed random values in the range between -0.1 and $+0.1$. If the initial weights are too large, the hidden units are saturated, and the gradient is also very small. The initial values for the RBF-centres were obtained from a *k*-means algorithm and the widths from a nearest neighbour heuristic. All the simulations were carried out on a Sun SPARC server 10-GS with 128 MB RAM. The simulations described are performed using the epoch-based stochastic version of backpropagation, where the weights are updated after each epoch of three (randomly chosen) patterns in the training set. This version is different from the batch version, where the weights are updated after the gradients have accumulated over the whole training set, and to the pattern-based version, where the weights are updated after the presentation of each pattern. The supervised learning minimised the standard objective (error) function, the sum of square of the output errors. Training and testing sets were chosen as simple random sample in each stratum of the eight training sites.

4 Classification Results

4.1 Overall Results: Performance of the Neural Classifiers with a Fixed Hidden Layer Size

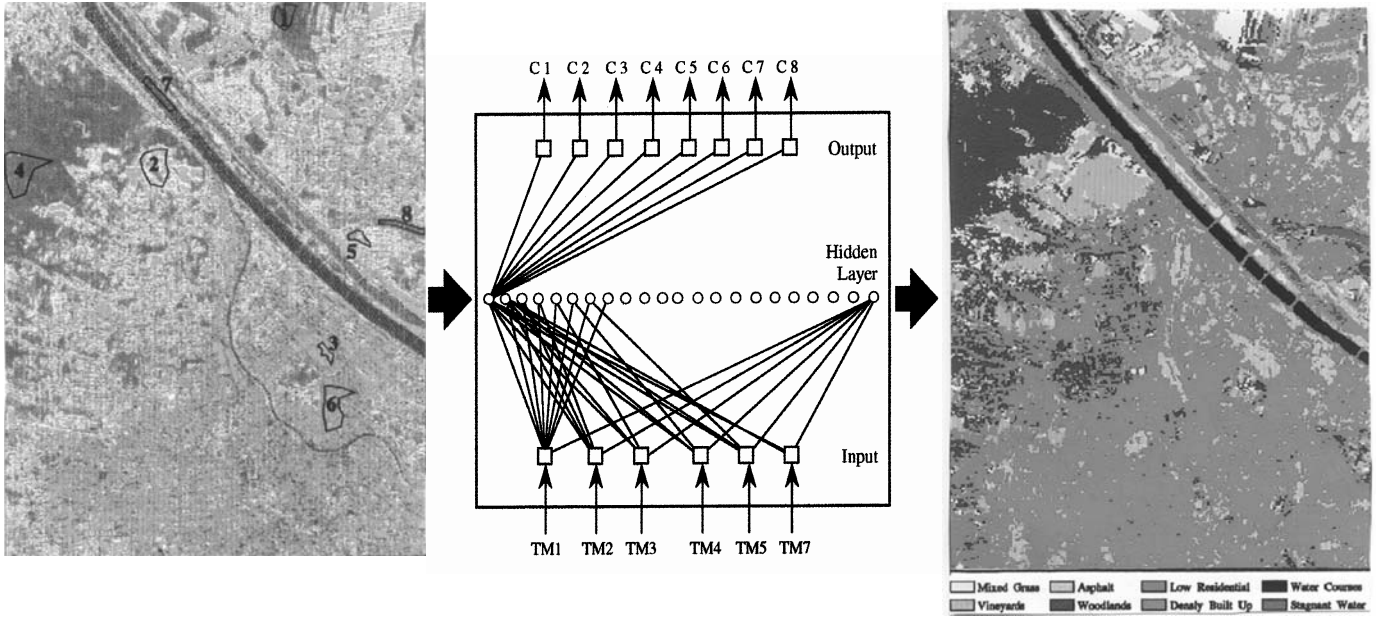
The purpose of the first experiment is to compare the in-sample and out-of-sample performance of the three neural classifiers each with 196 parameters, where the degrees of freedom are equal to 14. Thus, we were able to analyse the effect of

different hidden unit activation functions, the sigmoid (logistic), the hyperbolic tangent (tanh) and the radial basis activations, upon performance. All other factors including initial conditions are fixed in these simulations ($\eta = 0.8$). The results are outlined in Table 2 and show that the two MLP-classifiers trained more slowly than the RBF-classifier, but clearly outperform RBF (measured in terms of τ). The RBF-classifier does not train and generalise as accurately as the MLP-networks. Its results, however, strongly depend on the initial conditions for the RBF centres and widths. It is important to bear in mind that no attempts have been made here to optimise the results of this classifier with respect to these parameters. There seems to be much unexplored potential to improve the performance of this classifier. MLP-1 and MLP-2 generally train and generalise at the same rate, but MLP-1's training is faster, by about 30 percent.

Table 2 Summary of classification results

Classifier	*Classification accuracy* τ		*Convergence time*
	In-sample	*Out-of-sample*	(*CPU-time in sec.*)
MLP-1	92.13	89.76	15.1
MLP-2	90.91	90.00	21.0
RBF	80.00	75.61	10.6
NML	90.85	85.24	1.4

Thus, the best overall result is provided by the MLP-1 classifier with 14 hidden units and 196 free parameters, followed by MLP-2, and RBF. Both MLP classifiers outperform the NML classifier in terms of generalisation capabilities. The superiority of the MLP classifiers over RBF may be, moreover, underlined by considering the in-sample and out-of-sample classification error matrices (see Appendix B), the map user's and map producer's accuracies in Appendix C. Even though trained on 1,640 pixels only, the MLP-1 classifier can be used to classify the 97,200 pixels of the whole image. The raw satellite image and the MLP-1 classified image are displayed in Figure 4.



Raw Satellite Image:
City of Vienna and its Northern
Surrounding (Landsat 5 TM, June 5, 1985)

MLP-1 Classifier

MLP-1 Classified Image

Figure 4 The raw satellite and the MLP-1 classified image of the city of Vienna and its northern surroundings

4.2 Stability with Training Time

Figure 5 shows the in-sample performance for the two versions of the multi-layer perceptron, MLP-1 and MLP-2, and the radial basis function classifier as a function of training time in epochs (η = 0.8, trained for 50,000 epochs, and equal random initialisations). The in-sample performance tends to converge asymptotically at a minimum that is found at about 17,000 epochs in the case of the MLP-classifiers and about 36,000 epochs in the case of RFB.

There are some regions with temporary performance drops. At least, in the case of the MLP-classifiers we do not think that these can be interpreted as signs of overtraining, because they appear rather early in the training process. More probably, their existence implies that the network is still undertrained, and the better solutions are yet to come for larger numbers of epochs. This behaviour persists across the three different neural classifiers.

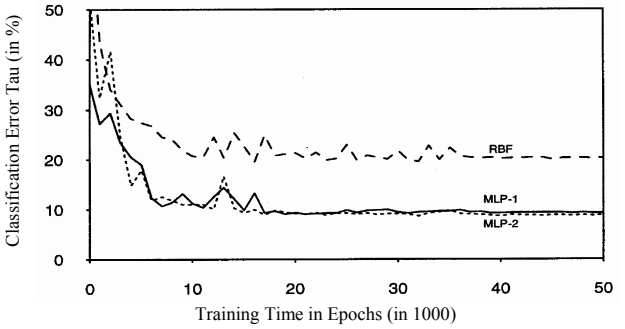


Figure 5 In-sample-performance of the MLP-1, MLP-2, and RBF classifiers (as a function of training time in epochs)

4.3 Stability with Initial Conditions

Backpropagation is known to be sensitive to the values of initial conditions of the parameters. The number of free parameters of MLP-1 is 196. The objective function has multiple local minima and is sensitive to details of initial values. A relatively small change in the initial values for the parameters generally results in finding a different local minimum. In this type of experiment we used three different sets of initial conditions. Initial weights were chosen from a uniform random distribution in (–0.1, +0.1).

Figure 6 shows the in-sample and out-of-sample classification error curves for the three trials. It is clear, that different initial conditions can lead to more or less major differences in the starting stage of the training process. After about 15,000 epochs the differences in performance more or less vanish. Nevertheless, it is important to stress that the issue of stability with initial conditions deserves consideration when training a classifier in a real world application context.

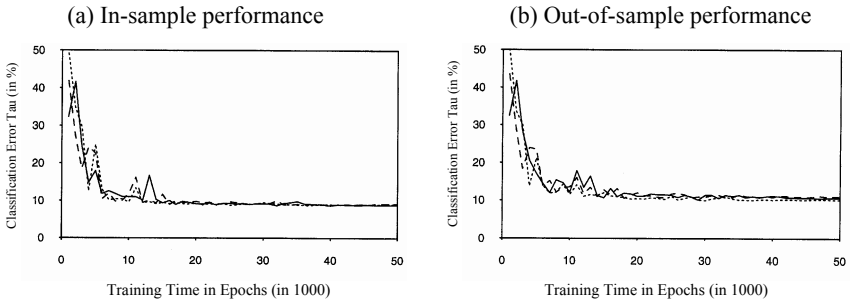


Figure 6 The effect of different initial parameter conditions on the performance of MLP-1

4.4 Stability with the Gradient Descent Control Term η

The choice of the control parameter for the gradient descent along the surface influences the magnitude of weight changes and, thus, is crucial for learning performance. But it is difficult to find appropriate learning rates. On one hand, a small learning rate implies small changes even though greater weight changes would be necessary. On the other, a greater learning rate implies greater weight changes. Greater weight changes might be required because of the speed of convergence on the network stability. Larger learning rate values might also assist the classifier to escape from a local minimum.

It is important to examine how the classification results vary with the gradient descent control term. A stability analysis with respect to this parameter shows that both in-sample and out-of-sample performance of the classifier remain very stable in the range of η = 0.4 to η = 0.8, while a small change from η = 0.4 to η = 0.2 yields a dramatic loss in classification accuracy (see Table 3). The optimal learning rate is the one which has the largest value that does not lead to oscillation, and this is η = 0.8 in this experiment. Figure 7 shows that a variable learning rate adjustment (declining learning rate: η = 0.8 until 5,000 epochs, η = 0.4 until 15,000 epochs, then η = 0.1 until 35,000 epochs and thereafter η = 0.00625) might lead to faster convergence, but only to a slightly better generalisation performance.

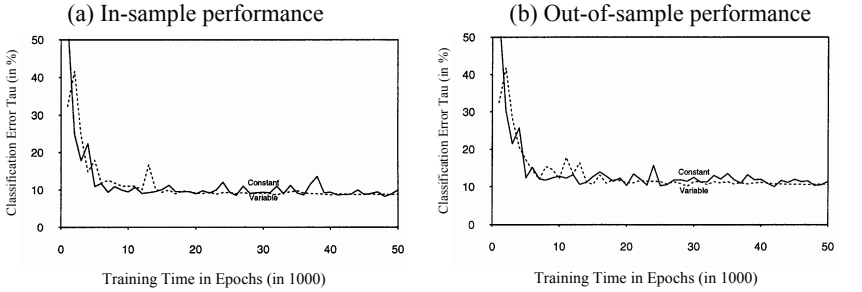


Figure 7 The effect of different approaches to learning rate adjustment on (a) in-sample performance and (b) out-of-sample performance of MLP-1: Constant (η = 0.8) versus variable learning rate adjustment

Table 3 Stability of results with the gradient descent control parameter as function of training time in epochs

Epochs ($x\ 10^3$)	*Control parameter* η	*In-sample performance* (*in terms of* τ)	*Out-of-sample performance* (*in terms of* τ)
3	0.2	16.6	12.5
	0.4	73.7	72.3
	0.6	78.2	78.5
	0.8	82.2	78.5
6	0.2	17.60	12.5
	0.4	90.17	88.2
	0.6	86.93	86.0
	0.8	88.28	84.9
9	0.2	21.56	12.5
	0.4	89.37	88.2
	0.6	90.22	87.5
	0.8	89.97	87.6
12	0.2	21.56	12.5
	0.4	88.37	85.4
	0.6	88.38	86.5
	0.8	90.92	86.8
15	0.2	22.54	12.7
	0.4	90.06	89.1
	0.6	88.93	87.9
	0.8	89.86	87.3
18	0.2	24.50	13.1
	0.4	89.55	87.3
	0.6	89.96	87.1
	0.8	90.51	88.5
21	0.2	24.50	13.1
	0.4	90.77	87.7
	0.6	91.48	88.3
	0.8	90.22	86.6
24	0.2	31.51	15.4
	0.4	91.47	88.2
	0.6	90.69	88.0
	0.8	87.87	84.3
27	0.2	31.51	15.4
	0.4	91.11	89.0
	0.6	89.96	87.2
	0.8	88.95	88.2
30	0.2	31.51	15.4
	0.4	90.81	89.2
	0.6	90.29	87.9
	0.8	90.59	87.5

4.5 Stability of Results with Different Training and Testing Samples

All the simulations we mentioned so far were performed for the same training and test data sets, obtained by stratified random sampling. To examine the effect of different training and test data sets on the performance, we used three randomly selected trials with stratification to generate training and testing sets of 1,640 and 820 pixels, respectively. In Figure 8 we see only minor differences. The in-sample performance of the classifier did not alter significantly after 15,000 epochs. The out-of-sample performance of two trials was rather similar after 36,000 epochs. However, one of the trials shows a different pattern in out-of-sample performance. If the training and test samples were randomly drawn without stratification, major differences in performance might arise between the trials (see Figure 9).

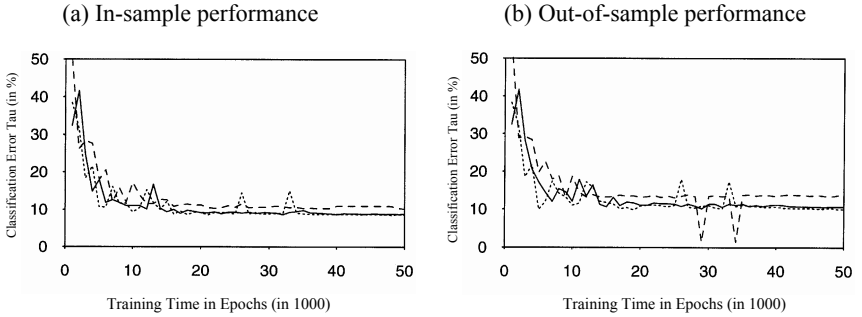


Figure 8 The effect of selected randomly chosen training/testing set trials **with** stratification on (a) in-sample performance and (b) out-of-sample performance of MLP-1 with variable learning rate adjustment

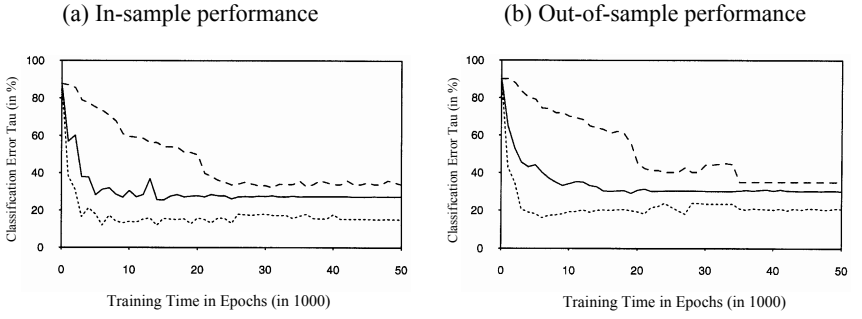


Figure 9 The effect of selected randomly chosen training/testing set trials **without** stratification on (a) in-sample performance and (b) out-of-sample performance of MLP-1 with variable learning rate adjustment

5 Conclusions

One major objective of this paper was to evaluate the classification accuracy of three neural classifiers, MLP-1, MLP-2 and RBF, and to analyse their generalisation capability and the stability of the results. We illustrated that both in-sample and out-of-sample performance depends upon fine-tuning of control parameters. Moreover, we were able to show that even a simple neural learning procedure such as the backpropagation algorithm outperforms by about five percent points the conventional classifier in generalisation that is most often used for multispectral classification on a pixel-by-pixel basis, the NML classifier. The nonlinear properties of the sigmoid (logistic) and the hyperbolic tangent (tanh) activation functions in combination with softmax activations of the output units allow neural network based classifiers to discriminate the data better and generalise significantly better, in the context of this study.

We strongly believe that with careful network design and multiple rather than single training sites and with a more powerful learning procedure, the performance of the neural network classifiers can be improved further, especially the RBF classifier. In this respect, other techniques than the *k*-means procedure might be more promising to use in order to obtain the initial values for the RBF centres and widths.

We hope that the issues addressed in this paper will be beneficial not only for designing neural classifiers for multispectral classification on a pixel-by-pixel basis, but also for other classification problems in the field of remote sensing, such as classification of multi-source data or multi-angle data.

***Acknowledgements*:** The authors gratefully acknowledge Professor Karl Kraus (Department of Photogrammetric Engineering and Remote Sensing, Vienna Technical University) for his assistance in supplying the remote sensing data used in this study. This work is supported by a grant from the Austrian Fonds zur Förderung der Wissenschaftlichen Forschung (P-09972-TEC), the US-National Science Foundation (SBR-930063), and NASA NAS5-31369.

References

Benediktsson J.A., Swain P.H. and Ersoy O.K. (1990): Neural network approaches versus statistical methods in classification of multisource remote sensing data, *IEEE Transactions on Geoscience and Remote Sensing* 28 (4), 540-551

Bischof H., Schneider W. and Pinz A.J. (1992): Multispectral classification of Landsat-images using neural networks, *IEEE Transactions on Geoscience and Remote Sensing* 30 (3), 482-490

Bridle J.S. (1990): Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: Fougelman-Soulié F. and Hérault J. (eds.) *Neuro-Computing: Algorithms, Architectures and Applications*, Springer, Berlin, Heidelberg, New York, pp. 227-236

Civco D.L. (1993): Artificial neural networks for land-cover classification and mapping, *International Journal of Geographical Information Systems* 7 (2), 173-186

Dreyer P. (1993): Classification of land cover using optimised neural nets on SPOT data, *Photogrammetric Engineering and Remote Sensing* 59 (5), 617-621

Fischer M.M. and Gopal S. (1994): Artificial neural networks: A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34 (4), 503-527

Gershenfield N.A. and Weigend A.S. (eds.) (1993): *Time Series Prediction*: *Forecasting the Future and Understanding the Past,* Addison-Wesley, Reading [MA]

Heerman P.D. and Khazenie N. (1992): Classification of multispectral remote sensing data using a backpropagation neural network, *IEEE Transactions on Geoscience and Remote Sensing* 30 (1), 81-88

Hepner G.F., Logan T., Ritter N. and Bryant N. (1990): Artificial neural network classification using a minimal training set: Comparison to conventional supervised classification, *Photogrammetric Engineering and Remote Sensing* 56 (4), 469-473

Key J., Maslanic A. and Schweiger A.J. (1989): Classification of merged AVHRR and SMMR Arctic data with neural networks, *Photogrammetric Engineering and Remote Sensing* 55 (9), 1331-1338

Lee J., Weger R.C., Sengupta S.K. and Welch R.M. (1990): A neural network approach to cloud classification, *IEEE Transactions on Geoscience and Remote Sensing* 28 (5), 846-855

Refenes A.N., Zapranis A. and Francis, G. (1994): Stock performance modelling using neural networks: A comparative study with regression models, *Neural Networks* 7 (2), 375-388

Rumelhart D.E., Hinton G.E. and Williams R.J. (1986): Learning internal representation by error propagation. In: Rumelhart D.E., McClelland J.L. and the PDP Research Group (eds.) *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, MIT Press, Cambridge [MA], pp. 318-362

Salu Y. and Tilton J. (1993): Classification of multispectral image data by the binary diamond neural network and by nonparametric, pixel-by-pixel methods, *IEEE Transactions on Geoscience and Remote Sensing* 31 (3), 606-617

Weigend A.S. (1993) Book Review: John A. Hertz, Anders S. Krogh and Richard G. Palmer, Introduction to the theory of neural computation, *Artificial Intelligence* 62, 93-111

Weigend A.S, Huberman, B.A. and Rumelhart, D.E. (1991): Predicting sunspots and exchange rates with connectionist networks. In: Eubank S. and Casdagli M. (eds.) *Proceedings of the 1990 NATO Workshop on Nonlinear Modelling and Forecasting*, Addison-Wesley, Redwood City [CA], pp. 1-36

White H. (1989): Some asymptotic results for learning in single hidden layer feedforward network models, *Journal of the American Statistical Association* 84 (408), 1003-1113

Wilkinson G.G., Fierens F. and Kanellopoulos I. (1995): Integration of neural and statistical approaches in spatial data classification, *Geographical Systems* 2 (1), 1-20

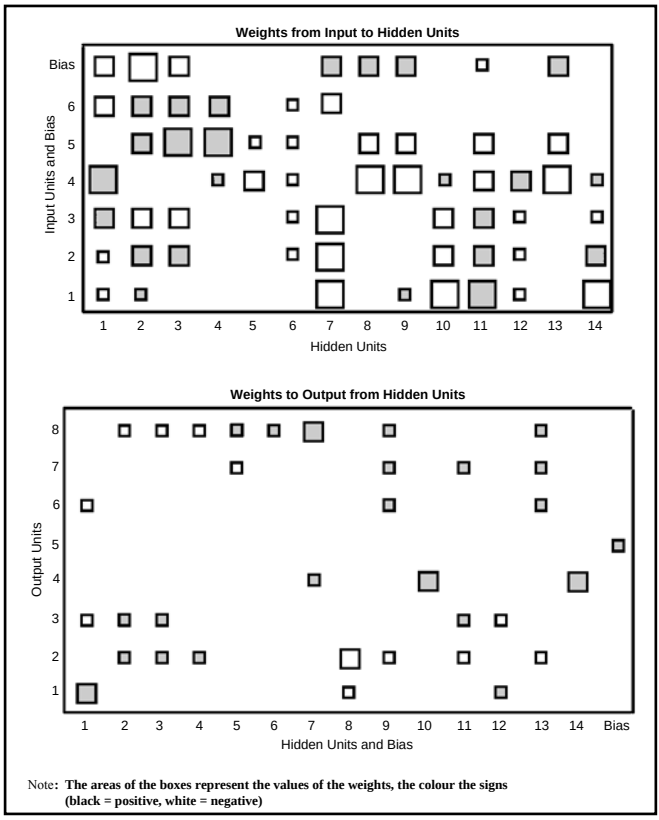


Figure A1 Weights of the MLP-1-classifier after weight elimination (17×10^3 epochs). Interpretation of these weights sheds light on which spectral channels are important for particular surface categories. Similarly, the connection weights indicate, for each output category, the degree of information redundancy among channels in the input data. Channels which are only weakly weighted add little additional information to the classification process. The identification of the exact role of the hidden units is difficult, as they often represent generalisations of the input patterns. Figure A1 shows with which input data channel each hidden node is associated in the trained network (top) and with which hidden unit each output class is related (bottom). The unit labelled 'bias' has output +1 and so represents the bias term. The areas of the boxes represent the values, the colour the signs (black = positive, white = negative). Following the connections through these two boxes, thus, indicates which input channels are linked to particular output categories

Appendix A: *Parameters of the MLP-1 Classifier after Weight Elimination*

The classifier was trained for 17,000 epochs with backpropagation and a constant learning rate of 0.8. The connection weights and biases of the network are given below in Table A1. When simulated serially on a SPARCserver 10-GS, the training took 15.1 CPU-minutes. Once the parameters have been determined, predictions are extremely fast.

Table A1 Weights of the MLP-1 classifier after weight elimination (17 x 10^3 epochs)

Weights from	*Input unit* 1		*Input unit* 2		*Input unit* 3		*Input unit* 4		*Input unit* 5		*Input unit* 6		*Bias unit*	
	Initial	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*
to hidden unit 1	–0.2654	–4.2068	–0.1314	–3.5575	–0.2212	–5.9796	0.3784	18.6484	0.3578	–0.0732	0.0869	–8.5990	0.2003	–8.3436
to hidden unit 2	0.1594	3.0924	0.4070	5.8573	0.2456	5.3472	0.0073	–0.3563	0.2850	7.5201	0.2688	5.5217	0.1842	–10.4762
to hidden unit 3	0.0531	1.8249	0.3094	5.8297	0.2921	5.2104	0.1607	–0.1826	–0.0433	12.0609	0.3005	7.7699	0.4015	–9.0623
to hidden unit 4	0.1994	0.4149	0.0774	0.7208	–0.2140	0.2577	–0.2098	4.3713	0.0882	10.3742	0.1856	6.9198	–0.2269	–1.1575
to hidden unit 5	–0.1601	0.3377	–0.3399	–0.3897	0.1033	0.1062	–0.4065	–5.9631	–0.3450	–4.9175	0.3534	–0.2195	–0.1589	0.7615
to hidden unit 6	–0.0044	–1.3902	–0.2401	–2.8702	–0.1541	–3.1082	0.0318	–3.2401	–0.0878	–3.6032	0.1167	–2.4680	–0.0406	–0.4469
to hidden unit 7	–0.3718	–15.9215	–0.3073	–14.7156	0.1205	–11.5417	–0.1708	–0.6922	0.0414	–1.6728	–0.3274	–5.5068	0.0237	7.1074
to hidden unit 8	0.3438	7.9521	0.3005	1.2669	0.3396	1.7644	0.1011	–14.2799	0.0615	–7.5545	0.3542	–0.0473	0.1576	8.3054
to hidden unit 9	0.0437	2.4169	0.0576	0.5233	0.1545	0.9377	–0.0733	–15.7223	0.2209	–7.2733	0.0608	–0.8447	0.2412	6.7711
to hidden unit 10	–0.3722	–13.5282	–0.2948	–9.6025	0.0408	–6.4631	0.1526	3.2262	0.2933	–0.4754	–0.0902	–1.7938	–0.0380	–0.1079
to hidden unit 11	0.0069	12.3658	0.1574	9.6851	0.3950	8.1292	–0.3037	–8.5565	–0.3066	–6.0183	–0.1253	1.2778	0.1470	–3.8310
to hidden unit 12	0.0750	–3.5478	–0.0813	–2.2013	0.2052	–2.2538	–0.1013	7.9774	–0.0173	0.1960	0.2132	–0.3957	–0.1855	0.6626
to hidden unit 13	–0.0528	1.5297	0.1934	0.5204	–0.3384	0.2251	–0.1238	–15.4543	0.0033	–6.3996	0.3912	–0.0856	–0.3394	7.0218
to hidden unit 14	–0.1923	–12.4562	–0.1975	–8.4983	0.0904	–4.8614	–0.3738	2.8841	–0.0765	–0.0992	0.1709	–0.8173	–0.1162	0.0340

TableA1 (*ctd.*)

Weights to	*Output unit* 1		*Output unit* 2		*Output unit* 3		*Output unit* 4		*Output unit* 5		*Output unit* 6		*Output unit* 7		*Output unit* 8	
	Initial	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*	*Initial*	*Final*
from hidden unit 1	0.0296	7.6672	–0.1638	0.1116	–0.0154	–2.1449	0.0306	1.3780	–0.1633	–0.2221	–0.1745	–2.0476	0.1451	–1.2741	–0.1667	–0.6374
from hidden unit 2	0.0313	–0.3342	0.0924	2.7388	–0.1079	3.7358	–0.0066	–1.3653	0.0776	0.7831	0.1954	0.0913	–0.0674	–0.5317	–0.1269	–2.1005
from hidden unit 3	–0.0727	–0.4291	0.0232	4.5698	0.2100	2.4571	–0.0060	–1.2327	0.1543	1.1922	0.1215	–0.0193	–0.2042	–1.1493	–0.1241	–2.6817
from hidden unit 4	0.1569	0.2916	0.1540	2.9387	0.1912	0.4205	0.0663	0.0305	–0.0894	0.3293	–0.2090	–0.5576	–0.0788	–2.0048	0.1540	–2.7553
from hidden unit 5	–0.1165	–0.5384	–0.0406	–1.1709	0.0888	v0.2694	–0.1419	–0.5978	0.1315	–0.0037	–0.0932	0.2617	–0.1675	0.5696	0.1175	2.2892
from hidden unit 6	0.0895	0.0044	–0.0086	–0.3923	0.0505	–0.4396	–0.0500	0.0434	–0.0742	–0.0712	–0.0528	0.0778	0.0694	0.2861	0.1096	2.6504
from hidden unit 7	–0.1257	0.2249	–0.1955	–1.7252	–0.1904	–1.7241	0.1733	4.7005	0.1701	0.1637	0.0223	–0.4417	–0.1298	–0.1447	0.0983	6.7245
from hidden unit 8	–0.1848	–3.3815	–0.1404	–6.3693	–0.1170	0.3268	–0.1681	–0.9975	0.0472	–0.1072	0.0691	1.6311	0.2088	1.8163	0.0129	0.8942
from hidden unit 9	–0.1908	–1.6496	–0.1914	–3.2548	0.1779	0.0828	–0.1485	–0.8071	0.0005	–0.2282	–0.0661	4.1493	0.2081	3.6822	0.1130	2.2155
from hidden unit 10	0.0026	1.0520	–0.1997	–1.3107	–0.0280	–1.1155	0.1611	6.2207	–0.1762	–0.3579	–0.0653	–1.1377	0.1879	–0.4309	–0.2095	–0.4046
from hidden unit 11	–0.0783	–1.0328	0.1830	–2.4157	0.1434	4.4601	–0.0311	–1.0268	–0.1831	–0.2165	0.0293	0.7224	0.1890	3.0847	–0.1837	0.0306
from hidden unit 12	0.1461	2.1664	–0.0339	0.4392	–0.0863	–4.0775	0.0292	0.4963	–0.0933	0.2816	–0.1719	–0.5003	–0.1109	–0.5863	–0.0690	–0.4879
from hidden unit 13	–0.2124	–1.6818	–0.1096	–2.9845	0.0001	0.0142	–0.1844	–0.7775	–0.0521	–0.2195	0.1410	4.0468	0.1640	3.6123	0.0173	2.0987
from hidden unit 14	0.0992	0.5098	–0.0655	–0.4612	–0.0141	–1.2092	0.2045	5.7548	0.0216	0.0802	–0.2110	–0.4714	–0.0020	–0.2936	–0.0036	0.1820
from bias unit	0.1249	0.2723	0.1453	0.7989	0.1157	–0.4729	0.1530	–0.2961	0.0318	3.4917	–0.1816	0.1615	–0.0615	–0.3898	0.1679	–0.6033

Appendix B: In-Sample and Out-of-Sample Classification Error Matrices of the Classifiers

An error matrix is a square array of numbers set out in rows and columns which expresses the number of pixels assigned to a particular category relative to the actual category as verified by some reference (ground truth) data. The columns represent the reference data, the rows indicate the classification generated. It is important to note that differences between the map classification and reference data might be not only due to classification errors. Other possible sources of errors include errors in interpretation and delineation of the reference data, changes in land cover between the data of the remotely sensed data and the data of the reference data (temporal error), variation in classification of the reference data due to inconsistencies in human interpretation etc.

Table B1 In-sample performance: Classification error matrices (f_{lk}) of the neural and the statistical classifiers

(*a*) *MLP-1*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	157	10	0	0	0	0	0	0	167
*C*2	1	282	0	0	2	0	0	0	285
*C*3	0	0	128	0	0	0	0	0	128
*C*4	4	0	0	389	9	0	0	0	402
*C*5	0	0	2	2	98	0	0	0	102
*C*6	0	0	1	0	0	260	25	10	296
*C*7	0	0	0	0	0	60	93	0	153
*C*8	0	0	0	0	0	3	0	104	107
Total	162	292	131	391	109	323	118	114	1,640

(*b*) *MLP-2*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	156	9	0	2	0	0	0	0	167
*C*2	4	280	0	1	0	0	0	0	285
*C*3	0	0	126	2	0	0	0	0	128
*C*4	4	0	0	384	14	0	0	0	402
*C*5	0	0	2	4	96	0	0	0	102
C6	0	0	1	0	0	253	25	17	296
*C*7	0	0	0	0	0	60	93	0	153
*C*8	0	0	0	0	0	4	0	103	107
Total	164	289	129	393	110	317	118	120	1,640

(*c*) *RBF*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	141	22	0	4	0	0	0	0	167
*C*2	14	263	0	4	0	4	0	0	285
*C*3	0	0	115	13	0	0	0	0	128
*C*4	9	0	0	349	44	0	0	0	402
*C*5	0	0	12	12	78	0	0	0	102
*C*6	0	0	5	0	0	189	71	31	296
*C*7	0	0	0	0	0	73	80	0	153
*C*8	0	0	0	0	0	10	0	97	107
Total	164	285	132	382	122	276	151	128	1,640

(*d*) *NML*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	161	5	0	1	0	0	0	0	167
*C*2	0	284	0	0	1	0	0	0	285
*C*3	0	0	124	0	4	0	0	0	128
*C*4	0	4	0	385	13	0	0	0	402
*C*5	0	0	0	0	102	0	0	0	102
*C*6	0	0	3	0	0	214	62	17	296
*C*7	0	0	0	0	0	37	116	0	153
*C*8	0	0	0	0	0	3	0	104	107
Total	161	293	127	386	120	254	178	121	1,640

Table B2 Out-of-sample classification error matrices (f_{lk}) of the neural and the statistical classifiers

(*a*) *MLP-1*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	79	4	0	0	0	0	0	0	83
*C*2	1	134	6	0	1	0	0	0	142
*C*3	0	0	64	0	0	0	0	0	64
*C*4	3	2	0	194	1	0	0	0	200
*C*5	0	3	0	0	49	0	0	0	52
*C*6	0	0	0	0	0	115	30	3	148
*C*7	0	0	0	0	0	29	48	0	77
*C*8	0	0	0	0	0	1	0	53	54
Total	83	143	70	194	51	145	78	56	820

(*b*) *MLP-2*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	79	4	0	0	0	0	0	0	83
*C*2	1	140	0	0	1	0	0	0	142
*C*3	0	0	64	0	0	0	0	0	64
*C*4	1	2	0	193	3	0	0	1	200
*C*5	0	1	0	0	51	0	0	0	52
*C*6	0	0	0	0	2	110	32	4	148
*C*7	0	0	0	0	0	29	48	0	77
*C*8	0	0	0	0	0	1	0	53	54
Total	81	147	64	193	57	140	80	58	820

(*c*) *RBF*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	35	21	0	27	0	0	0	0	83
*C*2	5	137	0	0	0	0	0	0	142
*C*3	0	4	60	0	0	0	0	0	64
*C*4	24	6	4	163	0	0	0	3	200
*C*5	0	26	4	0	22	0	0	0	52
*C*6	0	0	0	0	0	104	35	9	148
*C*7	0	0	0	0	0	29	48	0	77
*C*8	0	0	0	0	0	0	3	51	54
Total	64	194	68	190	22	133	86	63	820

(*d*) *NML*

Classifier's categories	*Ground truth categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	80	3	0	0	0	0	0	0	83
*C*2	0	141	0	0	0	0	0	0	142
*C*3	0	0	62	0	1	1	0	0	64
*C*4	1	3	0	191	5	0	0	0	200
*C*5	0	5	0	0	47	0	0	0	52
*C*6	0	0	1	0	2	73	64	8	148
*C*7	0	0	0	0	0	24	53	0	77
*C*8	0	0	0	0	0	2	0	52	54
Total	81	152	63	191	56	100	117	60	820

Appendix C: *In-Sample and Out-of-Sample Map User's and Map Producer's Accuracy*

Table C1 In-sample classification accuracy π and υ for the pattern classifiers

Category	*Name*	*Map user's accuracy* π				*Map producer's accuracy* υ			
		MLP-1	*MLP-2*	*RBF*	*NML*	*MLP-1*	*MLP-2*	*RBF*	*NML*
*C*1	Rural landscape	94.0	93.4	84.4	96.4	96.9	95.1	86.0	95.8
*C*2	Vineyards	98.9	98.3	92.3	99.7	96.6	96.9	92.3	96.9
*C*3	Asphalt	100.0	98.4	89.8	96.9	97.7	97.7	87.1	97.7
*C*4	Woodland	96.8	95.5	86.8	95.8	99.5	97.7	91.4	97.7
*C*5	Low residential	96.1	94.1	76.5	100.0	89.9	87.3	63.9	87.3
*C*6	Densely built up	87.8	85.5	63.9	72.3	80.5	79.8	68.5	79.8
*C*7	Water courses	60.8	60.8	52.3	75.8	78.8	78.8	53.0	78.8
*C*8	Stagnant water	97.2	96.3	90.7	97.2	91.2	85.8	75.8	85.8

Table C2 Out-of-sample classification accuracy π and υ for the pattern classifiers

Category	*Name*	*Map user's accuracy* π				*Map producer's accuracy* υ			
		MLP-1	*MLP-2*	*RBF*	*NML*	*MLP-1*	*MLP-2*	*RBF*	*NML*
*C*1	Rural landscape	95.2	95.2	42.2	96.4	95.2	97.5	54.7	98.8
*C*2	Vineyards	94.4	98.6	96.5	99.3	93.7	95.2	70.6	92.8
*C*3	Asphalt	100.0	100.0	93.8	96.9	91.4	100.0	88.2	98.4
*C*4	Woodland	97.0	96.5	81.5	95.5	100.0	100.0	85.8	100.0
*C*5	Low residential	94.2	98.1	42.3	90.4	96.1	89.5	100.0	83.9
*C*6	Densely built up	77.7	74.3	70.3	49.3	79.3	78.6	78.2	73.0
*C*7	Water courses	62.3	62.3	62.3	68.8	61.5	60.0	55.8	45.3
*C*8	Stagnant water	98.2	98.1	94.4	96.3	94.6	91.4	81.0	86.7

10 Optimisation in an Error Backpropagation Neural Network Environment with a Performance Test on a Spectral Pattern Classification Problem

with *P. Staufer*

This paper attempts to develop a mathematically rigid framework for minimising the cross-entropy function in an error backpropagating framework. In doing so, we derive the back-propagation formulae for evaluating the partial derivatives in a computationally efficient way. Various techniques of optimising the multiple-class cross-entropy error function to train single hidden layer neural network classifiers with softmax output transfer functions are investigated on a real world multispectral pixel-by-pixel classification problem that is of fundamental importance in remote sensing. These techniques include epoch-based and batch versions of backpropagation of gradient descent, PR-conjugate gradient, and BFGS quasi-Newton errors. The method of choice depends upon the nature of the learning task and whether one wants to optimise learning for speed or classification performance. It was found that, comparatively considered, gradient descent error backpropagation provided the best and most stable out-of-sample performance results across batch and epoch-based modes of operation. If the goal is to maximise learning speed and a sacrifice in classification accuracy is acceptable, then PR-conjugate gradient error backpropagation tends to be superior. If the training set is very large, stochastic epoch-based versions of local optimisers should be chosen utilising a larger rather than a smaller epoch size to avoid unacceptable instabilities in the classification results.

1 Introduction

Spectral pattern classification represents an area of considerable current interest and research. Satellite sensors record data in a variety of spectral channels and at a variety of ground resolutions. The analysis of remotely sensed data is usually achieved by machine-oriented pattern recognition techniques of which classification based on maximum likelihood, assuming Gaussian distribution of the data, is the most widely used one. As the complexity of satellite data grows, so too does the need for new tools to analyse them in general. Since the mid-1980s, neural network models have raised the possibility of realising fast, adaptive systems for multispectral satellite data classification.

Most applications of neural networks have involved image classification, generally with considerable success. A variety of data sources have been analysed with these models, including Landsat Multispectral Scanner (MSS) (see, for exam-

ple, Benediktsson et al. 1990, 1993), Landsat Thematic Mapper (TM) (see, McClellan et al. 1989, Hepner et al. 1990, Bischof et al. 1992, Heerman and Khazenie 1992, Civco 1993, Yoshida and Omatu 1994, Paola and Schowengerdt 1995, 1997, Bruzzone et al. 1997, Fischer et al. 1997, Skidmore et al. 1997), SPOT (Systeme pour l'Observation de la Terre) (see, Kanellopoulos et al. 1992, Chen et al. 1995), and synthetic aperture radar (SAR) (see, Foody 1995; Foody et al. 1995). Most of these studies rely on spectral reflectance properties of land cover types and operate on a pixel-by-pixel basis alone.

These supervised learning classification studies typically rely on feedforward neural networks, such as single hidden layer networks. Current practice is predominantly based on minimising the least squares error function in an error backpropagating environment, using backpropagation of gradient descent errors, introduced and popularised by Rumelhart et al. (1986), that is, a combination of the backpropagation technique for calculating the partial derivatives of the error function and the gradient descent procedure for updating the network parameters.

This contribution departs from current practice in two respects. First, we utilise the multiple-class cross-entropy error function rather than the least squares error function. In fact, the least squares error function is not the most appropriate for classification problems. It is motivated and derived from the principle of maximum likelihood on the assumption of the target data to be generated are from a smooth deterministic function with added Gaussian noise. However, the target values for a 1-of-C coding scheme (involving mutually exclusive classes) are binary, and, the Gaussian noise model does not provide a good description of their distribution. The multiple-class cross-entropy error function motivated from the concept of entropy – originally developed by physicists in the context of equilibrium thermodynamics – and paired with softmax output transfer functions ensures the network output values to be interpreted as probabilities of class membership and is likely to perform better than sum-of-squares at estimating small probabilities (Bishop 1995).

Second, most previous applications employed backpropagation of gradient descent errors or heuristic modifications for training neural networks. But the performance of backpropagation training can be greatly influenced by the choice of the optimisation procedure for parameter adaptation. Up to now the performance of other, more sophisticated optimisation procedures for parameter adjustment is unknown.

The purpose of this contribution is to develop a mathematically rigid framework for minimising the cross-entropy function in an error backpropagation environment. In doing so, we derive the backpropagation formulae, for evaluating the partial derivatives in a computationally efficient way. In addition we evaluate the efficacy of backpropagation training with three optimisation procedures, for weight updating, the gradient descent (GD), the one-step Polak-Ribiere-conjugate gradient (PR-CG); the Broyden-Fletcher-Goldfarb-Shanno (BFGS) memoryless quasi-Newton algorithms, on a multispectral pattern classification problem, with a challenging level of complexity and a larger size of training set. Two versions of off-line backpropagation training are considered: epoch-based learning (with epoch sizes k = 30, 300, 600, 900) and batch learning. They differ in how often

the weights are updated. The batch version updates the weights after all patterns have been propagated through the network. The epoch-based version updates using information from K^* patterns randomly chosen from the training set. The evaluation is based on three performance indices: learning time (measured in both as the elapsed time in CPU-seconds and the number of iterations) required to convergence on a solution, out-of-sample classification performance (measured in terms of total classification accuracy) and stability (that is, the ability of a network model with a given set of weights to converge, starting from different positions in parameter space).

The organisation of the paper is as follows. In the next section a summarised description of single hidden layer networks as classifiers is reviewed. Additionally, the network training problem is described as a problem of minimising the multiple-class cross-entropy error function. Training algorithms, whether used in off-line (epoch-based and batch) or on-line mode, are termed backpropagation algorithms if they use the technique of backpropagation for the evaluation of the error function derivatives (see Section 2) and some optimisation scheme for parameter updating. They basically differ in the choice of the optimisation procedure for weight adjustment. The three optimisation procedures used in the simulations in this study are discussed in Section 3: the gradient descent, the PR-conjugate gradient and the BFGS quasi-Newton algorithms. These techniques are compared on a supervised multispectral pixel-by-pixel classification problem in which the classifier is trained with examples of the classes to be recognised in the data set. This is achieved by using limited ground survey information that specifies where examples of specific categories are to be found in the satellite imagery. The multispectral pattern classification problem is characterised in Section 4 along with a discussion of the relative strengths and weaknesses of the above optimisation techniques when applied to solve the problem in batch and in epoch-based mode of operation. It will be demonstrated that the method of choice depends upon whether one wants to optimise learning speed or classification performance. If the training set is very large, stochastic epoch-based rather than deterministic batch modes of operations tend to be preferable.

2 Single Hidden Layer Networks and the Network Training Problem

Suppose we are interested in approximating a classification function $\mathcal{F}: \mathcal{R}^N \rightarrow \mathcal{R}^C$ which estimates, the probability that a pattern belongs to one of a priori known mutually exclusive classes. The function $\mathcal{F}$ is not analytically known but rather samples with $S = (s^1, ..., s^k)$ with $s^k = (\boldsymbol{x}^k, \boldsymbol{y}^k)$ are generated by a process that is governed by $\mathcal{F}$, that is, $\mathcal{F}(\boldsymbol{x}^k) = \boldsymbol{y}^k$. From the available samples we want to build a smooth approximation to $\mathcal{F}$. Note that in real world applications, only a finite (that is, small) number K of learning examples is available or can be used at the same time. Moreover, the samples contain noise.

To approximate $\mathfrak{F}$ we consider the class of single hidden layer feedforward networks Φ, the leading case of neural network models. Φ consists of a combination of transfer functions φ_j (j = 1, …, J) and ψ_c (c = 1, ..., C) that are represented by hidden units, and weighted forward connections between the input, hidden, and output units. The cth output element of Φ is

$$\Phi(\boldsymbol{x}, \boldsymbol{w})_c = \psi_c \left(\sum_{j=0}^{J} w_{cj} \; \varphi_j \left(\sum_{n=0}^{N} w_{jn} \; x_n \right) \right) \qquad 1 \le c \le C, \tag{1}$$

where N denotes the number of input units, J the number of hidden and C the number of output elements (a priori given classes). $\boldsymbol{x} = (x_0, x_1, ..., x_N)$ is the input vector augmented with a bias signal x_0 that can be thought of as being generated by a 'dummy' unit (with index zero) whose output is clamped at 1. The w_{jn} represent input to hidden connection weights, and the w_{cj} hidden to output weights (including the biases). The symbol $\boldsymbol{w}$ is a convenient short hand notion of the $\omega = [J(N + 1) + C(J + 1)]$-dimensional vector of all the w_{jn} and w_{cj} network weights and biases (that is, the model parameters). $\varphi_j(\cdot)$ and $\psi_c(\cdot)$ are differentiable non-linear transfer (activation) functions of, respectively, the hidden units (j = 1, ..., J) and the output elements (c = 1, ..., C).

One of the major issues in neural network modelling includes the problem of selecting an appropriate member of model class Φ in view of a particular real world application. This model specification problem involves both the choice of appropriate transfer functions $\varphi_j(\cdot)$ and $\psi_c(\cdot)$, and the determination of an adequate network topology of Φ. Clearly, the model choice problem and the network training problem (that is, the determination of an optimal set of model parameters where optimality is defined in terms of an error function) are intertwined in the sense that if a good model specification can be found, the success of which depends on the particular problem, then the step of network training may become easier to perform.

In this contribution, however, we restrict our scope to the network training problem and to training algorithms that train a fixed (that is, predetermined) member of class Φ. The approximation Φ to $\mathfrak{F}$ then only depends on the learning samples, and the learning (training) algorithm that determines the parameters $\boldsymbol{w}$ from S and the model specification. Let us assume the hidden unit transfer functions $\varphi_j(\cdot)$ to be identical, $\varphi_j(\cdot) = \varphi(\cdot)$ for all j = 1, …, J, and the logistic function. Thus, the output of the jth hidden unit, denoted by z_j reads as

$$z_j = \varphi(net_j) = \frac{1}{1 + \exp(-net_j)} \qquad j = 1, ..., J \tag{2}$$

with net_j representing the net input given by

$$net_j = \sum_{n=0}^{N} w_{jn}\ x_n \qquad j = 1, ..., J. \tag{3}$$

Moreover, each unit $c (c = 1, \ldots, C)$ of the output layer is assumed to have the same transfer function, denoted by $\psi(\cdot)$. Since the network should implement a classifier with real valued outputs, a generalisation of the logistic function known as softmax transfer function (Bridle 1990) represents an appropriate choice. With this specification the cth network output *is*

$$\Phi_c = \psi(net_c) = \frac{\exp(net_c)}{\sum_{c'=1}^{C} \exp(net_{c'})} \qquad c = 1, ..., C \tag{4}$$

with net_c representing the net input given by

$$net_c = \sum_{j=0}^{J} w_{cj}\ z_j = \sum_{j=0}^{J} w_{cj}\ \varphi(net_j) \qquad c = 1, ..., C. \tag{5}$$

This choice of this output transfer function is motivated by the goal of ensuring that the network outputs can be interpreted as probabilities of class membership, conditioned on the outputs z_j ($j = 1, ..., J$) of the hidden units (see Bishop 1995, p. 238 for more details).

The process of determining optimal parameter values is called training or learning and may be formulated in terms of the minimisation of an appropriate error function. The function that is minimised for the classification problem of this study is the multiple-class cross-entropy error function that is – following Bishop (1995, p. 238) – defined (for batch learning) as a sum over all training patterns and all outputs as

$$\begin{aligned} E(\boldsymbol{w}) &= \sum_{k=1}^{K} E^k(\boldsymbol{w}) = -\sum_{k=1}^{K}\sum_{c=1}^{C} y_c^k\ \ln\left(\frac{\Phi(\boldsymbol{x}^k, \boldsymbol{w})_c}{y_c^k}\right) \\ &= -\sum_{k=1}^{K}\sum_{c=1}^{C} y_c^k\ \ln\left\{\frac{1}{y_c^k}\ \frac{\exp\left[\sum_j w_{cj}(1+\exp(-\sum_n w_{jn}\ x_n))^{-1}\right]}{\sum_{c'} \exp\left[\sum_j w_{c'j}(1+\exp(-\sum_n w_{jn}\ x_n))^{-1}\right]}\right\}. \end{aligned} \tag{6}$$

Φ_c represents the cth component of the actual network output as a function of $\boldsymbol{x}^k$ and the weight vector $\boldsymbol{w}$, and may be interpreted as the network's estimate of the class membership probability. y_c^k is the target data which has a 1-of-C coding scheme so that $y_c^k = \delta_{cc'}$ for a training pattern $\boldsymbol{x}^k$ from class c' where $\delta_{cc'}$ denotes the Kronecker symbol with $\delta_{cc'} = 1$ for $c = c'$ and $\delta_{cc'} = 0$ otherwise. $E(\boldsymbol{w})$ can be calculated with one forward pass and the gradient $\nabla E(\boldsymbol{w})$ that is defined as

$$\nabla E(\boldsymbol{w}) = \left(..., \sum_{k=1}^{K} \frac{\partial E^k(\boldsymbol{w})}{\partial w_{jn}}, ..., \sum_{k=1}^{K} \frac{\partial E^k(\boldsymbol{w})}{\partial w_{cj}}, ... \right) \tag{7}$$

where K is the number of patterns presented to the network model during training, and $E^k(\boldsymbol{w})$ the local cross-entropy error associated with pattern k. Optimal weights $\boldsymbol{w}^* = (..., w_{jn}^*, ..., w_{cj}^*, ...)$ are obtained when $\Phi(\boldsymbol{x}^k)_c = \boldsymbol{y}_c^k$ for all $c = 1, ..., C$ and $k = 1, ..., K$. Characteristically, there exist many minima all of which satisfy

$$\nabla E(\boldsymbol{w}) = 0 \tag{8}$$

where $\nabla E(\boldsymbol{w})$ denotes the gradient of the error function in the ω-dimensional parameter space. The minimum for which the value of $E(\boldsymbol{w})$ is smallest is termed the global minimum while other minima are called local minima. But there is no guarantee about what kind of minimum is encountered. The problem is usually tackled by repeated application of the learning procedure from different starting configurations.

There are two basic approaches to find the minimum of the global error function E, off-line learning and on-line learning. They differ in how often the weights are updated. The *on-line* (that is, pattern-based) *learning* updates the weights after every single pattern s^k chosen at random from S, that is, using only information from one pattern. In contrast, *off-line learning* updates the weights after K^* patterns randomly chosen from S have been propagated through the network, that is, using information from K^* patterns in the training set. If $K^* = K$ off-line learning is known as *batch learning,* otherwise it is also termed *epoch-based learning* with an epoch size of K^* ($1 < K^* < K$).

Both, the on-line and epoch-based (K^* small) versions are not consistent with optimisation theory, but nevertheless have been found to be superior to batch learning on real world problems that show a realistic level of complexity and have a training set that goes beyond a critical threshold (see Le Cun 1989, Schiffmann et al. 1993).

3 Optimisation Strategy and the Backpropagation Technique

In the previous section the network training problem has been formulated as a problem of minimising the multiple-class cross-entropy error function, and, thus, as a special case of function approximation where no explicit model of the data is assumed. Most of the optimisation procedures used to minimise functions are based on the same strategy. The minimisation is a local iterative process in which an approximation to the function in a neighbourhood of the current point in parameter space is minimised. The approximation is often given by a first-order or second-

order Taylor expansion of the function. In the case of batch learning, the general scheme of the iteration process may be formulated as follows:

(i) choose an initial vector $\boldsymbol{w}$ in parameter space and set $\tau = 1$,

(ii) determine a search direction $\boldsymbol{d}(\tau)$ and step size $\eta(\tau)$ so that

$$E(\boldsymbol{w}(\tau) + \eta(\tau)\, \boldsymbol{d}(\tau)) < E(\boldsymbol{w}(\tau)) \qquad \tau = 1, 2, ... \tag{9}$$

(iii) update the parameter vector

$$\boldsymbol{w}(\tau + 1) = \boldsymbol{w}(\tau) + \eta(\tau)\, \boldsymbol{d}(\tau) \qquad \tau = 1, 2, ... \tag{10}$$

(iv) if $dE(\boldsymbol{w})/d\boldsymbol{w} \neq 0$, then set $\tau = \tau + 1$ and go to (ii), else return $\boldsymbol{w}(\tau + 1)$ as the desired minimum.

In the case of *on-line learning* the above scheme has to be slightly modified since this learning approach is based on the (local) error function E^k, and the parameter vector $\boldsymbol{w}^k(\tau)$ is updated after every presentation of $s^k = (\boldsymbol{x}^k, \boldsymbol{y}^k)$. In both cases, batch and on-line learning, determining the next current point in the iteration process is faced with two problems. First, the search direction $\boldsymbol{d}(\tau)$ has to be determined, that is, in what direction in parameter space do we want to go in the search for a new current point. Second, once the search direction has been found, we have to decide how far to go in the specified direction, that is, a step size $\eta(\tau)$ has to be determined. For solving these problems characteristically two types of operations have to be accomplished: first, the computation or the evaluation of the derivatives of the error function (6) with respect to the network parameters, and, second, the computation of the parameter $\eta(\tau)$ and the direction vector $\boldsymbol{d}(\tau)$ based upon these derivatives.

In the sequel we illustrate that the backpropagation technique is a powerful method for efficiently calculating the gradient of the cross-entropy error function (6) with respect to the parameter weights. Because the single hidden layer network is equivalent to a chain of function compositions [see Equation (1)], the chain rule of differential calculus will play a major role in finding the gradient of function (6). In order to keep the notation uncluttered we will omit the superscript k (representing the kth training pattern) from the terms, except the error function.

Let us consider, first, the partial derivatives of $E^k(\boldsymbol{w})$ with respect to the hidden-to-output connection parameters, that is, the w_{cj} weights with $c = 1, ..., C$ and $j = 1, ..., J$. Note that $E^k(\boldsymbol{w})$ depends on w_{cj} only via the summed input net_c to output unit c [cf. Equation (5)]. Thus, using the chain rule for differentiation one may express the partial derivatives of $E^k(\boldsymbol{w})$ we are looking for as (for training pattern k)

$$\frac{\partial E^k}{\partial w_{cj}} = \frac{\partial E^k}{\partial net_c} \frac{\partial net_c}{\partial w_{cj}}. \tag{11}$$

Recall from Equation (5) that $net_c = \Sigma_{j=1}^{J} w_{cj}\ z_j$ where the sum is taken over the output of all hidden units j = 1, ..., J. Thus, the second term of the right-hand side of Equation (11) can be evaluated as follows:

$$\frac{\partial net_c}{\partial w_{cj}} = \frac{\partial}{\partial w_{cj}} \sum_{j=1}^{J} w_{cj}\ z_j = \frac{\partial}{\partial w_{cj}} \left[\sum_{j' \neq j} w_{cj'}\ z_{j'} + w_{cj}\ z_j \right] = z_j. \tag{12}$$

Substituting (12) into (11) we obtain

$$\frac{\partial E^k}{\partial w_{cj}} = \frac{\partial E^k}{\partial net_c}\ z_j. \tag{13}$$

If we define the local error at the cth node of the network, called delta, by

$$\delta_c := \frac{\partial E^k}{\partial net_c}, \tag{14}$$

we can express the partial derivatives of E with respect to w_{cj} as

$$\frac{\partial E^k}{\partial w_{cj}} = \delta_c\ z_j. \tag{15}$$

Equation (15) tells us that the required partial derivative $\partial E^k / \partial w_{cj}$ is obtained simply by multiplying the value of δ_c, associated with the output end of the connection parameter w_{cj}, with the value of z_j, associated with the input end of the connection parameter w_{cj}. Note that $z_0 = 1$. Thus, in order to evaluate the partial derivatives in question we need only to calculate the value of δ_c for each c = 1, ..., C and then apply (15).

This leads to the task to evaluate δ_c. Once again applying the chain rule we obtain

$$\delta_c = \frac{\partial E^k}{\partial net_c} = \sum_{c'=1}^{C} \frac{\partial E^k}{\partial \Phi_{c'}} \frac{\partial \Phi_{c'}}{\partial net_c}. \tag{16}$$

From (6) we have

$$\frac{\partial E^k}{\partial \Phi_{c'}} = \frac{\partial}{\partial \Phi_{c'}}\left[-\sum_{c''=1}^{C} y_{c''} \ln\left[\frac{\Phi_{c''}}{y_{c''}}\right]\right] = -\frac{y_{c'}}{\Phi_{c'}} \tag{17}$$

and from (4)

$$\frac{\partial \Phi_{c'}}{\partial net_c} = \frac{\partial}{\partial net_c}\left[\frac{\exp(net_c)}{\sum_{c'''} \exp(net_{c'''})}\right]$$

$$= \delta_{cc'} \frac{\exp(net_c)}{\sum_{c'''} \exp(net_{c'''})} - \frac{\exp(net_{c'})\exp(net_c)}{\left(\sum_{c'''} \exp(net_{c'''})\right)^2} = \partial_{cc'}\, \Phi_c - \Phi_c\, \Phi_{c'} \tag{18}$$

where $\partial_{cc'}$ is the usual Kronecker symbol. Substituting (17) and (18) into (1) leads to

$$\delta_c = \sum_{c'=1}^{C}\left[-\frac{y_{c'}}{\Phi_{c'}}\right][\delta_{cc'}\, \Phi_c - \Phi_c\, \Phi_{c'}] = \sum_{c \neq c'} y_{c'}\, \Phi_c - y_c + y_c\, \Phi_c$$

$$= \left[\sum_{c \neq c'} y_{c'} + y_c\right]\Phi_c - y_c = \Phi_c - y_c. \tag{19}$$

Thus, the partial derivative we are looking for is

$$\frac{\partial E^k}{\partial w_{cj}} = \delta_c\, z_j = (\Phi_c - y_c) z_j. \tag{20}$$

Let us consider now the second set of partial derivatives, $\partial E^k / \partial w_{jn}$ for $j = 1, ..., J$ and $n = 1, \ldots, N$. This is a little more complicated. Again we apply the chain rule for differentiation and finally arrive at

$$\frac{\partial E^k}{\partial w_{jn}} = \delta_j\, x_n \tag{21}$$

with

$$\delta_j = \varphi'(net_j)\sum_{c=1}^{C} w_{cj}\, \delta_c. \tag{22}$$

Note that the local error of the hidden units is determined on the basis of the local errors at the output layer [δ_c is given by (19)]. The chain rule gives

$$\frac{\partial E^k}{\partial w_{jn}} = \frac{\partial E^k}{\partial z_j} \frac{\partial z_j}{\partial net_j} \frac{\partial net_j}{\partial w_{jn}} \tag{23}$$

with

$$\frac{\partial net_j}{\partial w_{jn}} = \frac{\partial}{\partial w_{jn}} \sum_{n=1}^{N} w_{jn}\, x_n = \frac{\partial}{\partial w_{jn}} \left[\sum_{n' \neq n} w_{jn'}\, x_{n'} + w_{jn}\, x_n \right] = x_n \tag{24}$$

$$\delta_j := \frac{\partial E^k}{\partial z_j} \frac{\partial z_j}{\partial net_j} \tag{25}$$

where the second term of the right-hand side of Equation (25) is evaluated as

$$\frac{\partial z_j}{\partial net_j} = \varphi'(net_j) = \varphi(net_j)(1 - \varphi(net_j)) \tag{26}$$

and the first term as

$$\frac{\partial E^k}{\partial z_j} = \frac{\partial}{\partial z_j} \left\{ \sum_{c=1}^{C} y_c \ln \left[\frac{1}{y_c} \frac{\exp\left(\sum_j w_{cj}\, z_j\right)}{\sum_{c'} \exp\left(\sum_j w_{c'j}\, z_j\right)} \right] \right\}$$

$$= \sum_{c=1}^{C} w_{cj} \left[\Phi_c - y_c \right] = \sum_{c=1}^{C} w_{cj}\, \delta_c . \tag{27}$$

Substituting (26)—(27) into *(25)* leads to (22).

In summary, the backpropagation technique for evaluating the partial derivatives of the cross-entropy error function with respect to the w_{cj} and the w_{jn} parameters can be described with three major steps:

Step 1: *Feedforward computation*: Select (preferably at random) a training pattern $(\boldsymbol{x}^k, \boldsymbol{y}^k)$ from the set of training samples and propagate $\boldsymbol{x}^k$ forward through the network using Equations (2)-(5), thus generating the hidden and output unit activations based on current weight settings.

Step 2: *Set of partial derivatives* $\partial E^k / \partial w_{cj}$: Compute the δ_c for all the output units c = 1, ..., C using (19), and utilise (20) to evaluate the required derivatives.

Step 3: *Set of partial derivatives* $\delta E^k / \delta w_{jn}$: Backpropagate the deltas using (22) and (19) backward to the hidden layer to obtain δ_j for each hidden

unit $j = 1, \ldots, J$ in the network model, and utilise (21) to evaluate the required derivatives.

In *on-line learning* the error function gradient is evaluated for just one pattern at a time, and the parameters updated using (11) where the different patterns $s^k = (\boldsymbol{x}^k, \boldsymbol{y}^k)$ in the training set S can be considered in sequence, or more typically selected at random. For *off-line* (that is, batch and epoch-based) *learning,* the derivative of the total error $E(\boldsymbol{w})$ can be obtained by repeating the above three steps for each training sample s^k, and then summing over all samples, that is,

$$\frac{\partial E}{\partial w_{cj}} = \sum_{k=1}^{K^*} \frac{\partial E^k}{\partial w_{cj}} \tag{28}$$

and

$$\frac{\partial E}{\partial w_{jn}} = \sum_{k=1}^{K^*} \frac{\partial E^k}{\partial w_{jn}} \tag{29}$$

with $K^* = K$ in the case of batch learning, and $K \gg K^* > 1$ in the case of epoch-based learning with epoch size K^*. It is worthwhile to note that in the stochastic version of epoch-based learning K^* training patterns are chosen randomly in each iteration step of the optimisation strategy.

4 Optimisation Techniques for Parameter Adjustments

In numerical optimisation different techniques for the computation of the parameter $\eta(\tau)$ and the direction vector $\boldsymbol{d}(\tau)$ are known (see, for example, Luenberger 1984, Fletcher 1986). We consider three techniques that are used for the multispectral pattern classification task to be described in Section 4:

(i) The *steepest-descent* (*gradient*) *method* (GD) defines the direction as the negative gradient

$$\boldsymbol{d}(\tau) := -\nabla E(w(\tau)) \qquad \tau = 1, 2, \ldots \tag{30}$$

(ii) *Conjugate gradient* (CG) *methods* calculate the actual search direction $\boldsymbol{d}(\tau)$ as a linear combination of the gradient vector and the previous search directions (see Hestenes and Stiefel 1952). In the PR-CG algorithm, the Polak-Ribière variant of conjugate gradient procedures (see Press et al. 1992), that is used in the simulations presented in Section 4, the search direction is computed as

$$\boldsymbol{d}(\tau) := -\nabla E(\boldsymbol{w}(\tau)) + \beta(\tau)\,\boldsymbol{d}(\tau-1) \qquad \tau = 1, 2, \ldots \tag{31}$$

with

$$\boldsymbol{d}(0) := -\nabla E(\boldsymbol{w}(0)) \tag{32}$$

where $\beta(\tau)$ is a scalar parameter that ensures that the sequence of vectors $\boldsymbol{d}(\tau)$ satisfy the following condition expressed as

$$\beta(\tau) = \frac{\left[\nabla E(\boldsymbol{w}(\tau)) - \nabla E(\boldsymbol{w}(\tau-1))\right]^{\mathrm{T}}\ \nabla E(\boldsymbol{w}(\tau))}{\nabla E(\boldsymbol{w}(\tau-1))^{\mathrm{T}}\ \nabla E(\boldsymbol{w}(\tau-1))}. \tag{33}$$

$\boldsymbol{w}(\tau-1)^{\mathrm{T}}$ is the transpose of $\boldsymbol{w}(\tau-1)$. Note that the CG algorithm utilises information about the direction search $\boldsymbol{d}(\tau-1)$ from the previous iteration in order to accelerate convergence, and each search direction would be conjugate if the objective function would be quadratic.

(iii) *Quasi-Newton* – also called variable *metric – methods* employ the differences of two successive iteration points τ and $\tau+1$, and the difference of the corresponding gradients to approximate the inverse Hessian matrix. The most commonly used update technique is the BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm (see Luenberger 1984, Fletcher 1986, Press et al. 1992) that determines the search direction as

$$\boldsymbol{d}(\tau) = -\boldsymbol{H}(\tau)\ \nabla E(\boldsymbol{w}(\tau)) \tag{34}$$

where $\boldsymbol{H}(\tau)$ is some (ω, ω)-dimensional symmetric positive definite matrix and denotes the current approximation to the inverse of the Hessian matrix, that is,

$$\boldsymbol{H}(\tau) \cong \left[\nabla^2 E(\boldsymbol{w}(\tau))\right]^{-1} \tag{35}$$

where

$$\boldsymbol{H}(\tau) = \left\{\boldsymbol{I} - \frac{\boldsymbol{d}(\tau-1)(\boldsymbol{g}(\tau-1))^{\mathrm{T}}}{(\boldsymbol{d}(\tau-1))^{\mathrm{T}}\,\boldsymbol{g}(\tau-1)}\right\}\boldsymbol{H}(\tau-1)\left\{\boldsymbol{I} - \frac{\boldsymbol{g}(\tau-1)(\boldsymbol{d}(\tau-1))^{\mathrm{T}}}{(\boldsymbol{d}(\tau-1))^{\mathrm{T}}\,\boldsymbol{g}(\tau-1)}\right\} + \frac{\boldsymbol{d}(\tau-1)(\boldsymbol{d}(\tau-1))^{\mathrm{T}}}{(\boldsymbol{d}(\tau-1))^{\mathrm{T}}\boldsymbol{d}(\tau-1)} \tag{36}$$

with

$$\boldsymbol{g}(\tau-1) := \nabla E(\boldsymbol{w}(\tau)) - \nabla E(\boldsymbol{w}(\tau-1)). \tag{37}$$

$\boldsymbol{H}$ is initialised usually with the identity matrix $\boldsymbol{I}$ and updated at each iteration using only gradient differences to approximate second-order information. The inverse Hessian is more closely approximated as iterations proceed.

Both the PR-CG and the BFGS algorithms raise the calculation complexity per training iteration considerably since they have to perform a one-dimensional linear search in order to determine an appropriate step size. A line search involves several calculations of either the global error function E or its derivative, both of which raise the complexity. Characteristically, the parameter $\eta = \eta(\tau)$ is chosen to minimise

$$E(\tau) = E(\boldsymbol{w}(\tau)) + \eta\, \boldsymbol{d}(\tau) \tag{38}$$

in the τth iteration. This gives an automatic procedure for setting the step length, once the search direction $\boldsymbol{d}(\tau)$ has been determined.

All these three procedures use only first-order derivative information of the error function: The derivatives can, thus, be calculated efficiently by backpropagation as shown in Section 2. The steepest descent algorithm has the great advantage of being very simple and cheap to implement. One of its limitations is the need to choose a suitable step size η by trial and error. Inefficiency is primarily due to the fact that the minimisation directions and step sizes may be poorly chosen. Unless the first step is chosen such that it leads directly to the minimum, steepest descent will zigzag with many small steps. It is worth noting that there have been numerous attempts in recent years to improve the performance of the basic gradient descent by making various ad hoc modifications (see, for example, Jacobs 1988), such as the heuristic scheme known as *quick-prop* (Fahlman 1988).

In contrast, the conjugate gradient and quasi-Newton procedures are intrinsically off-line parameter adjustment techniques, and evidently more sophisticated optimisation procedures. In terms of complexity and convergence properties, the conjugate gradient can be regarded as being somewhat intermediate between the method of gradient descent and the quasi-Newton technique (Cichocki and Unbehauen 1993). Its advantage is the simplicity for estimating optimal values of the coefficients $\eta(\tau)$ and $\beta(\tau)$ at each iteration. No (ω, ω)-dimensional matrices $\boldsymbol{H}(\tau)$ need to be generated as in the quasi-Newton procedures. The search direction is chosen by appropriately setting the β so that $\boldsymbol{d}$ distorts as little as possible the minimisation achieved by the previous search step. A major difficulty is that for the non-quadratic, error function (6) the obtained directions are *not* necessarily descent directions and numerical instability can result (Battiti and Tecchiolli 1994). Periodically, it might be necessary to restart the optimisation process by a search in the steepest descent directon when a non-descent search direction is generated. It is worthwhile to mention that the conjugate gradient procedures can be viewed as a form of gradient descent with an adaptive momentum $\beta(\tau)$, the im-

portant difference being that $\eta(\tau)$ and $\beta(\tau)$ in conjugate gradient are automatically determined at each iteration [see Equations (33) and (38)].

But the conjugate gradient methods are not as effective as some quasi-Newton procedures. It is noteworthy that quasi-Newton techniques are a shortcut for the Newton-Raphson algorithm that speeds up computations when the derivative calculation is time consuming. They require approximately twice as many gradient evaluations as the quasi-Newton methods. However, they save time and memory required for calculating the (ω, ω)-dimensional matrices $\boldsymbol{H}(\tau)$, especially in the case of large-sized classification problems (Shanno 1990). In quasi-Newton procedures, the matrices $\boldsymbol{H}(\tau)$ are positive definite approximations of the inverse Hessian matrix obtained from gradient (first-order) information. Thus, it is not required to evaluate second-order derivatives of E. A significant advantage of the quasi-Newton over the conjugate gradient procedures is that line search needs not to be performed with such a great accuracy since it does not form a critical feature in the algorithms. It is worthwhile to mention that scaled conjugate gradient algorithms provide a means of avoiding time-consuming line search of conventional conjugate gradients by utilising the model-trust region approach, known from the Levenberg-Marquardt algorithm, a variation of the standard Newton algorithm [see Møller (1993) for more details]. The algorithms such as BFGS are always stable since $\boldsymbol{d}(\tau)$ is always a descent search direction. They are today the most efficient and sophisticated optimisation techniques for batch training. But they are expensive both in computation and memory. Large-sized real world classification problems implying larger ω could lead to prohibitive memory requirements (Shanno 1990).

5 The Multispectral Pattern Classification Task and Test Results

The following training procedures that represent the major classes of the optimisation techniques are compared:

(i) GD: error backpropagation with gradient descent minimisation and with fixed and constant step sizes (that is, the standard backpropagation technique that is most widely used in geoscientific application contexts);

(ii) PR-CG: error backpropagation with the Polak-Ribiere conjugate gradient minimisation;

(iii) BFGS: error backpropagation with the Broyden-Fletcher-Goldfarb-Shanno quasi-Newton minimisation

in batch mode as well as in epoch-based operation with epoch sizes K^* = 30, 300, 600, 900.

These procedures are compared on a supervised multispectral pixel-by-pixel classification problem in which the classifier is trained with examples of the classes to be recognised in the data set. This is achieved by using limited ground survey information that specifies where examples of specific categories are to be found in the satellite imagery. Such ground truth information has been gathered on sites which are well representative of the larger area analysed from space. The image data set consists of 2,460 pixels (resolution cells) selected from a Landsat Thematic Mapper (TM) scene (270 x 360 pixels) from the city of Vienna and its northern surroundings (observation date: June 5, 1985; location of the center: 16°23'*E*, 48°14'*N*; TM Quarter Scene 190-026/4). The six Landsat TM spectral bands used are blue (SB1), green (SB2), red (SB3), near infrared (SB4), and mid-infrared (SB5 and SB7), excluding the thermal band with only a 120-meter ground resolution. Thus, each TM pixel represents a ground area of 30 m x 30 m and has six spectral band values varying over 256 digital numbers (8 bits).

Table 1 Classes and number of training/testing pixels

	Description	*Pixels*	
		Training	*Testing*
Class c_1	Mixed grass and arable land	167	83
Class c_2	Vineyards and areas with low vegetation cover	285	142
Class c_3	Asphalt and concrete surfaces	128	64
Class c_4	Woodland and public gardens	402	200
Class c_5	Low density suburban areas	102	52
Class c_6	Densely built up urban areas	296	148
Class c_7	Water courses	153	77
Class c_8	Stagnant water	107	54
Total number of pixels		1,640	820

The purpose of the multispectral classification task at hand is to distinguish between the eight classes of urban land use listed in Table 1. The classes chosen are meaningful to photo interpreters and land-use managers, but are not necessarily spectrally homogeneous. This classification problem used to evaluate the classification performance of the above training procedures in a real world context is challenging. The pixel-based remotely sensed spectral band values are noisy and sometimes unreliable. Some of the urban land-use classes are sparsely distributed in the image. The number of training sites is small relative to the number of land-use categories (one site training case). The training sites vary between 154 pixels (class suburban) and 602 pixels (class woodland and public gardens with trees). The above mentioned six TM bands provide the data set input for each pixel, with values scaled to the interval [0.1, 0.9]. This approach resulted in a database consisting of 2,460 pixels (about 2.5 percent of all the pixels in the scene) that are described by six-dimensional feature vectors, each tagged with its correct class membership. The set was divided into a training set (two thirds of the training site

pixels) and a testing set by stratified random sampling – stratified in terms of the eight classes. Pixels from the testing set are not used during network training and serve only to evaluate out-of-sample classification accuracy when the trained classifier is presented with novel data. The goal is to predict the correct class category for the test sample of pixels. In remote sensing classification tasks classification accuracy can be more important than fast learning.

A single hidden layer network classifier, with $N = 6$ input units (representing the six-dimensional feature vectors), $J = 14$ hidden units and $C = 8$ output units (representing the eight urban land use classes), was used in the study along with logistic hidden and softmax output transfer functions. Using a pruning technique $J = 14$ has been found in a previous study (Fischer et al. 1997) to control the effective complexity of the network (complexity in terms of the model parameters). The network was initialised with random weights in the range [–0.1, 0.1]. Weights were updated in batch and epoch-based modes. The latter with a range of four epoch sizes ($K^* = 30, 300, 600, 900$). Learning was stopped when every element of the gradient vector had an absolute value of less than 10^{-6}. This termination criterion is considered as a realistic test for convergence on a minimum (Shanno 1978). Otherwise, training was terminated if the number of iterations exceeded 100,000. The test set was presented at convergence or after a maximum of 100,000 iterations in order to monitor classification performance.

Each experiment (that is, combination of training algorithm with tuning parameters) was repeated ten times, the network classifier being initialised with a different set of random weights before each trial. To enable more accurate comparisons, the classifier was initialised with the same ten sets of random weights for all experiments. All experiments were done on a SUN Ultra 1 workstation. For implementation of PR-CG and BFGS, we used library algorithms (Press et al. 1992) modified as necessary to work in an error backpropagation neural network environment.

Table 2 presents the results for experiments involving batch mode of operation. Averages and standard deviations of performance indices were calculated only for those trials which converged within the 100,000 iterations limit. The times shown are CPU seconds in the sense that they exclude overheads such as scoring on the test set and screen display of progress information. In-sample (out-of-sample) performance is measured in terms of the percentage of training (testing) pixels correctly classified at convergence. In order to do justice to each algorithm, optimal combinations of parameters were systematically sought. GD requires time consuming tuning of the learning rate parameter η to get optimum performance. The possible restarts and line search combinations for PR-CG and BFGS also require tuning, particularly when using inexact searches. Since the error surface is often highly nonquadratic, it is important to use a line search that deals successfully with nonconvexities etc.

Although GD could quickly learn the training set, it could not find a local minimum in less than 6,499 iterations. Convergence to a minimum was impossible except with very low η-values which make convergence extremely slow. By contrast, PR-CG and BFGS proved to be much faster on average. This can also be

observed from the learning curves displayed in Figure 1. To avoid cluttering the graphs Figure 1 shows only the learning curves averaged over all the converged simulations of the ten trials. The GD-learning curve clearly illustrates the extremely slow convergence of GD-optimisation. In contrast, PR-CG and BFGS have the advantage of faster convergence. The rate of convergence for the PR-CG is significantly higher than that of BFGS. But the difference is not statistically significant. The higher convergence rate of both these techniques was offset by greater computational complexity in terms of CPU-time (see Table 2) and exhaustive memory requirements. In contrast to BFGS, RP-CG saves time and memory needed for computing a totally dense matrix $\boldsymbol{H}(\tau)$ at each iteration step τ, but requires approximately twice as many gradient evaluations.

Table 2 Batch learning: Comparative performance of error backpropagation with different optimisation techniques[a]

	Error backpropagation with		
	GD (η=0.0008)	*PR-CG*	*BFGS*
Number converged	9	10	8
Time	0.43 (0.67)	9.36 (12.49)	48.85 (44.00)
Iterations	19,443 (25,343.86)	8,306 (4,650.37)	9,582.75 (16,659.89)
Function values	263.35 (29.63)	110.271 (17.15)	244.00 (79.58)
In-sample classification accuracy	93.08 (0.82)	96.65 (0.42)	93.81 (1.63)
Out-of-sample classification accuracy	84.20 (1.83)	74.60 (2.26)	76.10 (3.77)

[a] Performance values represent the mean (standard deviation in brackets) of converged simulations. Number converged: Number of simulations converged within 100,000 iterations, out of ten trials differing in the initial random weights. The same ten sets of weights were used for each algorithm. Time: CPU seconds required to reach the convergence condition. Function value: Multiple-class cross-entropy error function value at convergence. In-sample-classification accuracy: Percentage of training pixels correctly classified at convergence. out-of-sample-classification accuracy: Percentage of test pixels correctly classified at convergence.

The average values of the multiple-class cross-entropy function do seem to indicate that PR-CG tended to find better local minima than any other procedure, and this conclusion is corroborated by the fact that the standard deviation after training in the ten runs is significantly lower as shown in Table 2. Especially, BFGS appears to be more prone to fall into local minima as indicated from the rather high standard deviation. Moreover, Table 2 clearly indicates that better out-of-sample classification performance is not the result of finding a lower minimum (see also Battiti and Tecchiolli 1994). PR-CG and BFGS seem to utilise information to modify the direction of steepest descent that resulted in significantly poorer out-of-sample classification accuracy on our task. In fact, GD outperforms PR-CG by 9.60 percentage points and BFGS by 8.10 percentage points on average. An interesting conclusion from this comparative study is that out-of-sample classification performance can vary between algorithms and even between different trials of

the same algorithm, despite all of them finding a local minimum. It is important to note that GD out-of-sample performance varies between 80.49 and 86.22 percent of classification accuracy, while PR-CG out-of-sample performance varies between 70.21 and 78.54, and BFGS between 67.80 and 79.88 percent only.

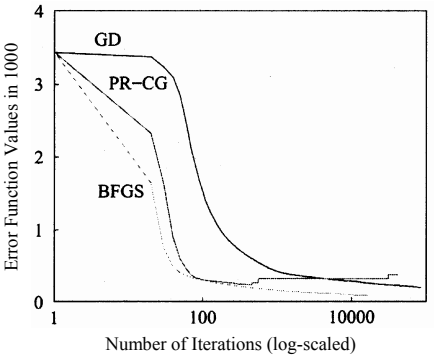


Figure 1 Batch learning curves as a function of training time: The effect of different optimisation techniques (averaged values of converged simulations)

The second series of experiments involves epoch-based rather than batch learning, with a range of epoch sizes K^* = 900, 600, 300 and 30. The results obtained are summarised in Table 3 along with the corresponding learning curves displayed in Figure 2. Epoch-based learning strategies may be more effective than batch learning, especially when the number K of training examples is very large and many training examples possess redundant information in the sense that many contributions to the gradient are very similar. Epoch-based updating makes the search path in the parameter space stochastic when the input vector is drawn at random. The main difficulty with stochastic epoch-based learning is its apparent inability to converge on a minimum within the 100,000 iterations limit.

While modifications to increase speed are important and attract most attention, classification accuracy is perhaps more important in applications such as pixel-by-pixel classification of remotely sensed data. Important differences in out-of-sample performance between batch and epoch-based learning may be observed. First, with larger K^* epoch-based learning tends to outperform batch out-of-sample classification accuracies. This is especially true for PR-CG optimisation. The best classification results for all three optimisation techniques are obtained for K^*= 900. The out-of-sample classification performance of PR-CG is improved by 8.92 percentage points in classification accuracy on average (at the expense of less stable solutions), and that of BFGS by 6.74 percentage points. GD produces slightly better and much more stable out-of-sample results. Second, better out-of-sample performance is not the result of finding lower multiple-class cross-entropy function values. Third, out-of-sample performance tends to slightly decrease with decreasing epoch size and at the same time the uncertainty in the generalisation obtained increases. This can be seen from the larger amplitude of the oscillation in

Figure 2(c) and especially Figure 2(d). Oscillation in results causes a large standard deviation. If high classification accuracy and stability are more important than faster learning, then GD error backpropagation exhibits superiority over PR-CG and BFGS independently of the mode. If, however, the goal is to maximise learning and a sacrifice in classification accuracy is acceptable, then conjugate gradient error backpropagation is the method of choice.

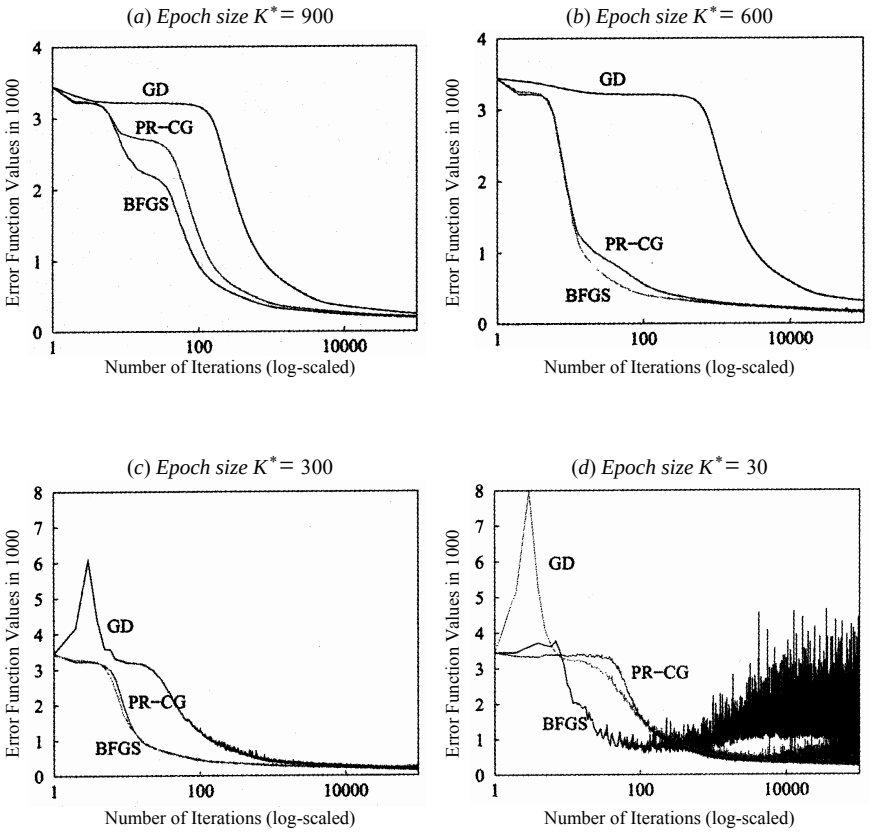


Figure 2 Epoch-based learning curves as a function of training time: The effect of different optimisation techniques (averaged values over ten trials)

Table 3 Epoch-based learning: Comparative performance of error backpropagation with different optimisation techniques[a]

	Error backpropagation with		
	GD (η = 0.0008)	*PR-CG*	*BFGS*
Epoch size K^* = 900 (GD: η = 0.0003)			
Time	21.97 (3.44)	28.14 (4.67)	42.74 (6.84)
Function values	249.01 (3.24)	210.16 (16.06)	196.05 (21.23)
In-sample classification accuracy	93.43 (0.17)	94.18 (0.59)	94.74 (0.82)
Out-of-sample classification accuracy	85.67 (0.73)	83.52 (4.38)	82.84 (3.85)
Epoch size K^* = 600 (GD: η = 0.0001)			
Time	21.88 (2.57)	23.45 (3.14)	37.91 (5.39)
Function values	312.76 (3.82)	167.05 (20.53)	151.86 (17.04)
In-sample classification accuracy	92.27 (0.08)	95.16 (0.59)	95.74 (0.53)
Out-of-sample classification accuracy	85.28 (1.07)	79.60 (5.42)	76.14 (4.92)
Epoch size K^* = 300 (GD: η = 0.008)			
Time	20.82 (1.98)	17.22 (4.33)	30.33 (4.95)
Function values	196.33 (7.67)	157.38 (11.68)	190.37 (29.34)
In-sample classification accuracy	94.87 (0.28)	95.41 (0.55)	95.38 (0.39)
Out-of-sample classification accuracy	83.56 (2.41)	77.50 (4.12)	76.12 (3.73)
Epoch size K^* = 30 (GD: η = 0.08)			
Time	17.10 (0.45)	11.37 (0.65)	20.24 (3.34)
Function values	247.58 (27.80)	790.47 (310.99)	1,451.83 (374.08)
In-sample classification accuracy	93.59 (1.20)	84.05 (9.14)	81.66 (5.02)
Out-of-sample classification accuracy	81.29 (3.17)	74.65 (6.84)	75.05 (4.40)

[a] Performance values represent the mean (standard deviation in brackets) of ten simulations differing in the initial random weights. Time: CPU seconds required to reach the convergence condition. Function value: Multiple-class cross-entropy function value after 10^5 iterations. In-sample classification accuracy: Percentage of training pixels correctly classified after 10^5 iterations. Out-of-sample-classification accuracy: Percentage of test pixels correctly classified after 10^5 iterations.

The superiority of GD error backpropagation in terms of the classification accuracy over PR-CG backpropagation is underlined by considering the out-of-sample classification error matrices, the commission and omission errors in Table 4 (epoch size K^* = 900). GD error backpropagation yielded slightly higher classification accuracies than PR-CC backpropagation for most of the landcover classes. Misclassification errors are unequally distributed among the landcover classes in both cases. In particular, class c_8 (water courses) was poorly classified by both optimisation procedures. This unequal distribution may be due to the difficulties inherent in the data set used, such as spectra complexities of the landcover classes, the presence of mixed pixels, and possible inaccuracies in the collection of the reference samples.

Table 4 Out-of-sample error matrices for error backpropagation with (a) gradient descent and (b) conjugate gradient minimisation [epoch size $K^* = 900$, best out of ten simulations differing in the initial random weight values]

(*a*) *Gradient descent error backpropagation*

Classifier's categories	*Ground truth categories*								*Row total*	*Commission error* [%]
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8		
c_1	75	1	0	7	0	0	0	0	83	9.64
c_2	2	139	0	1	0	0	0	0	142	2.11
c_3	1	0	60	0	2	1	0	0	64	6.25
c_4	1	2	0	196	0	0	0	1	200	2.00
c_5	0	4	0	0	48	0	0	0	52	7.69
c_6	0	0	0	0	1	139	1	7	148	6.18
c_7	0	0	0	0	0	76	1	0	77	98.70
c_8	0	0	0	0	0	1	0	53	54	1.85
Column total	79	146	60	204	51	217	2	61	820	–
Omission error [%]	5.06	4.79	0.00	3.92	5.88	35.94	50.00	13.11	–	–

(*b*) *Conjugate gradient error backpropagation*

Classifier's categories	*Ground truth categories*								*Row total*	*Commission error* [%]
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8		
c_1	73	1	0	9	0	0	0	0	83	12.05
c_2	5	135	0	1	1	0	0	0	142	4.93
c_3	1	0	62	0	0	1	0	0	64	3.12
c_4	1	2	0	194	3	0	0	0	200	3.00
c_5	0	6	0	0	46	0	0	0	52	11.64
c_6	0	0	0	0	1	141	3	3	148	4.73
c_7	0	0	0	0	0	75	1	1	77	98.70
c_8	0	0	0	0	0	0	0	54	54	0.00
Column total	80	144	62	204	51	217	4	58	820	–
Omission error [%]	8.75	6.25	0.00	4.90	9.80	35.02	75.00	6.90	–	–

6 Concluding Remarks

Backpropagation training is central to most geoscientific applications of neural networks. We have mathematically shown that the backpropagation technique is a powerful method for efficiently calculating the gradient of the cross-entropy error function, an important alternative to the widely used the least squares error function for pattern classification.

Most of the previous remote sensing applications employed backpropagation of gradient descent errors. In this contribution we viewed the process of neural network training as an optimisation process. More precisely, the training process is considered as a local iterative process in which an approximation to the multiple-class cross-entropy function in a neighbourhood of the current point in parameter space is minimised. This perspective opens the possibility to combine the backpropagation technique with more sophisticated optimisation procedures for parameter adjustment, the PR-CG and BFGS techniques. The performance of error backpropagation with gradient descent, Polak-Ribiere conjugate gradient, and Broyden-Fletcher-Goldfarb-Shanno quasi-Newton minimisation is evaluated and tested in the context of urban landcover classification.

The results obtained may be summarised as follows. First, the choice of the optimisation strategy (batch versus epoch-based mode of operation) and of the optimisation technique (GD, PR-CG, and BFGS) depends on the nature of the learning task and whether one wants to optimise learning for speed or classification performance. Second,, if the goal is to maximise learning speed on a pattern classification problem and a sacrifice in classification accuracy is acceptable, then PR-CG error backpropagation, the most mature technique, would be the method of choice. Third, where high classification accuracy and stability is more important than faster learning, then GD error backpropagation exhibits superiority over PR-CG and BFGS in view of our pixel-by-pixel pattern classification task – independently of the mode of operation – but requires time-consuming tuning of the learning parameter η to achieve 'best' out-of-sample performance. Fourth, if the training set is very large, stochastic epoch-based rather than deterministic batch modes of operation should be chosen, with a larger rather than a smaller epoch size. Much work on such optimisers, however, is still required before they can be utilised with the same confidence and ease with which batch local optimisers are currently used.

References

Battiti R. and Tecchiolli G. (1994): Learning with first, second, and no derivatives: A case study in high energy physics, *Neurocomputing* 6 (2), 181-206

Benediktsson J.A., Swain P.H. and Ersoy O.K. (1993): Conjugate-gradient neural networks in classification of multisource and very-high-dimensional remote sensing data, *International Journal of Remote Sensing* 14 (15), 2883-2903

Benediktsson J.A., Swain P.H. and Ersoy O.K. (1990): Neural network approaches versus statistical methods in classification of multisource remote sensing data, *IEEE Transactions on Geoscience and Remote Sensing* 28 (4), 540-551

Bischof H., Schneider W. and Pinz A.J. (1992): Multispectral classification of Landsat-images using neural networks, *IEEE Transactions on Geoscience and Remote Sensing* 30 (3), 482-490

Bishop C.M. (1995): *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford

Bridle J.S. (1990): Probabilistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition. In: Fogelman-Soulié F. and

Hérault J. (eds.) *Neurocomputing*: *Algorithms, Architectures, and Applications*, Springer, Berlin, Heidelberg, New York, pp. 227-236

Bruzzone L., Conese C., Maselli F. and Roli F. (1997): Multisource classification of complex rural areas by statistical and neural-network approaches, *Photogrammetric Engineering & Remote Sensing* 63 (5), 523-533

Chen K.S., Tzeng Y.C., Chen C.F. and Kao W.L. (1995): Land-cover classification of multispectral imagery using a dynamic learning neural network, *Photogrammetric Engineering & Remote Sensing* 61 (4), 403-408

Cichocki A. and Unbehauen R. (1993): *Neural Networks for Optimisation and Signal-Processing*, John Wiley, Chichester [UK], New York

Civco D.L. (1993): Artificial neural networks for land-cover classification and mapping, *International Journal for Geographical Information Systems* 7 (2), 173-186

Fahlman S.E. (1988): Faster-learning variations on backpropagation: An empirical study. In: Touretzky D., Hinton G.E. and Sejnowski T.J. (eds.) *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufman, San Mateo [CA], pp. 38-51

Fischer M.M., Gopal S., Staufer P. and Steinnocher K. (1997): Evaluation of neural pattern classifiers for a remote sensing application, *Geographical Systems* 4 (2), 195-224 and 243-244

Fletcher R. (1986): *Practical Methods for Optimization*, Wiley-Interscience, New York

Foody G.F. (1995): Land cover classification by a neural network with ancillary information, *International Journal for Geographical Information Systems* 9, 527-542

Foody G.F., McCulloch M.B. and Yates W.B. (1995): Classification of remotely sensed data by an artificial neural network: Issues related to the training set characteristics, *Photogrammetric Engineering & Remote Sensing* 61 (4), 391-401

Heerman P.D. and Khazenie N. (1992): Classification of multispectral remote sensing data using a backpropagation neural network, *IEEE Transactions on Geoscience and Remote Sensing* 30 (1), 81-88

Hepner G.F., Logan T., Ritter N. and Bryant N. (1990): Artificial neural network classification using a minimal training set: Comparison to conventional supervised classification, *Photogrammetric Engineering & Remote Sensing* 56 (4), 469-473

Hestenes M.R. and Stiefel E. (1952): Methods of conjugate gradients for solving linear systems, *Journal of Research of the National Bureau of Standards* 49 (6), 409-436

Jacobs R.A. (1988): Increased rates of convergence through learning rate adaptation, *Neural Networks* 1 (4), 295-307

Kanellopoulos I., Varfis A., Wilkinson G.G. and Megiér J. (1992): Landcover discrimination in SPOT HRV imagery using an artificial neural network – A 20-Class experiment, *Remote Sensing of the Environment* 13 (5), 917-924

Le Cun Y. (1989): Generalization and network design strategies. In: Pfeifer M. (ed.) *Connections in Perspective*, North-Holland, Amsterdam, pp. 143-155

Luenberger P. (1984): *Linear and Nonlinear Programming*, Addison-Wesley, Reading, [MA]

McClellan G.E., DeWitt R.N., Hemmer T.H., Mattheson L.N. and Moe G.O. (1989): Multispectral image processing with a three-layer backpropagation network. In: *International Joint Conference on Neural Networks*, IEEE Press, Piscataway [NJ], pp. I: 151-153

Møller M.F. (1993): A scaled conjugate gradient algorithm for fast supervised learning, *Neural Networks* 6 (4), 525-533

Paola J.D. and Schowengerdt R.A. (1995): A review and analysis of backpropagation neural networks for classification of remotely sensed multispectral imagery, *International Journal of Remote Sensing* 16 (16), 3033-3058

Paola J.D. and Schowengerdt R.A. (1997): The effect of neural-network structure on a multispectral land-use/land-cover classification, *Photogrammetric Engineering & Remote Sensing* 63 (5), 535-544

Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P. (1992): *Numerical Recipes in C. The Art of Scientific Computing*, Cambridge University Press, Cambridge [MA]

Rumelhart D.E., Hinton G.E. and Williams R.J. (1986): Learning internal representations by error propagation. In: Rumelhart D.E., McClelland L.J. and the PDP Research Group (eds.) *Parallel Distributed Processing*: *Explorations in the Microstructures of Cognition*, MIT Press, Cambridge [MA], pp. 318-332

Schiffmann W., Jost M. and Werner R. (1993): Comparison of optimised backpropagation algorithms. In: Verleysen M. (ed.) *European Symposium on Artificial Neural Networks*, Brussels, pp. 97-104

Shanno D.F. (1978): Conjugate gradient methods with inexact searches, *Mathematics of Operations Research* 3 (3), 244-256

Shanno D.F. (1990): Recent advances in numerical techniques for large-scale optimization. In: Miller W.T. (ed.) *Neural Networks for Robotics and Control*, MIT Press, Cambridge [MA], pp. 171-178

Skidmore A.K., Turner B.J., Brinkhof W. and Knoles E. (1997): Performance of a neural network: Mapping forests using GIS and remotely sensed data, *Photogrammetric Engineering & Remote* Sensing 63 (5), 501-514

Yoshida T. and Omatu S. (1994): Neural network approach to landcover mapping, *IEEE Transactions on Geoscience and Remote Sensing* 32, 1103-1109

List of Symbols Used

n	input unit label	w_{cj}	connection weight from hidden unit j to output unit c
j	hidden unit label	w_{jn}	connection weight from input unit n to hidden unit j
c	output unit (class) label	ω	dimension of the parameter space
N	number of input units	$\boldsymbol{d}$	search direction vector
J	number of hidden units	η	step size (scalar)
C	number of output units (classes)	τ	iteration step
K	number of learning patterns	E	total error function
K^*	epoch size	E^k	local error function
k	learning pattern label	∇	gradient
$\boldsymbol{x}^k$	kth learning input pattern	∂	partial derivative
$\boldsymbol{y}^k$	kth learning output pattern	δ_j	local error of the jth hidden node
ψ_c	transfer function of the cth output unit	δ_c	local error of the cth ouput node
φ_j	transfer function of the jth hidden unit	S	set of training patterns
φ'	derivative of φ	$\mathfrak{F}$	analytically unknown mapping from input to output space
exp	exponential function	$\in$	element of
ln	logarithm to base e	$:=$	defined as
$\mathfrak{R}$	space of real numbers	Φ	single hidden layer feedforward network function
$\boldsymbol{x}^T$	transpose of $\boldsymbol{x}$	Φ_c	cth elemtent of Φ
$\delta_{cc'}$	Kronecker symbol		
$\boldsymbol{w}$	vector of all w_{cj} and w_{jn} network weights and biases		
$\boldsymbol{w}^*$	vector of optimal w_{cj}^*- and w_{jn}^*-patterns		

ctd.

z_j	activation of the jth hidden unit	y_c	cth component of $\boldsymbol{y}$
net_j	net input to the jth hidden unit	x_0	bias signal
net_c	net input to the cth output unit	$\gg$	much greater than
x	N-dimensional vector, element of space $\mathcal{R}^N$	β	scalar parameater
		$\boldsymbol{H}$	Hessian matrix
y	C-dimensional vector, element of space $\mathcal{R}^C$	$\boldsymbol{I}$	identity matrix
		$\boldsymbol{g}$	gradient vector
x_n	nth component of $\boldsymbol{x}$	$\cong$	approximate

11 Fuzzy ARTMAP – A Neural Classifier for Multispectral Image Classification

with *S. Gopal*

This chapter shifts attention to fuzzy ARTMAP classification which synthesises fuzzy logic and Adaptive Resonance Theory (ART) by exploiting the formal similarity between the computations of fuzzy subsets and the dynamics of category choice, search and learning. The contribution describes design features, system dynamics and simulation algorithms of this learning system, which is trained and tested for classification (with eight classes a priori given) of a Landsat-5 Thematic Mapper scene from the city of Vienna on a pixel-by-pixel basis. The performance of the fuzzy ARTMAP is compared with that of an error-based learning system based upon a single hidden layer feedforward network, and the Gaussian maximum likelihood classifier as conventional statistical benchmark on the same database. Both neural classifiers outperform the conventional classifier in terms of classification accuracy. Fuzzy ARTMAP leads to out-of-sample classification accuracies which are very close to maximum performance, while the backpropagation network – like the conventional classifier – has difficulty in distinguishing between some land use categories.

1 Introduction

Spectral pattern recognition deals with classifications that utilise pixel-by-pixel spectral information from satellite imagery. The literature on neural network applications in this area is relatively new, dating back only about six to seven years. The first studies established the feasibility of error-based learning systems such as backpropagation (see Key et al. 1989, McClellan et al. 1989, Benediktsson et al. 1990, Hepner et al. 1990). Subsequent studies analysed backpropagation networks in more detail and compared them to standard statistical classifiers such as the Gaussian maximum likelihood (see Bischof et al. 1992, Kanellopoulos et al. 1993, Fischer et al. 1994).

In this chapter we analyse the capability and applicability of a different class of neural networks, called fuzzy ARTMAP, to multispectral image classification. Fuzzy ARTMAP synthesises fuzzy logic and Adaptive Resonance Theory (ART) models by describing the dynamics of ART category choice, search and learning in terms of analog fuzzy set-theoretic rather than binary set-theoretic operations. The chapter describes design features, system dynamics and simulation algorithms of this learning system, which is trained and tested for classification of a multi-

spectral image of a Landsat-5 Thematic Mapper (TM) scene (270x360 pixels) from the City of Vienna on a pixel-by-pixel basis. Fuzzy ARTMAP performance is compared with that of a backpropagation system and the Gaussian maximum likelihood classifier on the same database.

The chapter is organised as follows. Section 2 gives a brief mathematical description of the unsupervised learning system, called ART 1, which is a prerequisite to understanding the ARTMAP system. Section 3 shows how two ART 1 modules are linked together to form the ARTMAP supervised learning system for binary pattern recognition problems. Section 4 leads to one generalisation of ARTMAP, called fuzzy ARTMAP, that learns to classify continuous valued rather than binary patterns, and to a simplified version of the general fuzzy ARTMAP learning system, which will be used as general purpose remote sensing classifier in this study. Section 5 describes the remote sensing classification problem which is used to test the classifier's capabilities. The simulation results are given in Section 6 and compared with those obtained by the backpropagation network and the conventional maximum likelihood classifier. The final section contains some conclusions.

2 Adaptive Resonance Theory and ART 1

The basic principles of adaptive resonance theory (ART) were introduced by Stephen Grossberg in 1976 as a theory of human cognitive information processing (Grossberg 1976 a, b). Since that time the cognitive theory has led to a series of ART neural network models for category learning and pattern recognition. Such models may be characterised by a system of ordinary differential equations (Carpenter and Grossberg 1985, 1987a) and have been implemented in practice using analytical solutions or approximations to these differential equations.

ART models come in several varieties, most of which are unsupervised, and the simplest are ART 1 designed for binary input patterns (Carpenter and Grossberg 1987a) and ART 2 for continuous valued (or binary) inputs (Carpenter and Grossberg 1987b). This section describes the ART 1 model which is a prerequisite to understanding the learning system fuzzy ARTMAP. The main components of an ART 1 system are shown in Figure 1. Ovals represent fields (layers) of nodes, semicircles adaptive filter pathways and arrows paths which are not adaptive. Circles denote nodes (processors), shadowed nodes the vigilance parameter, the match criterion and gain control nuclei that sum input signals. The F_1 nodes are indexed by i and the F_2 nodes by j (categories, prototypes). The binary vector $\boldsymbol{I} = (I_1, ..., I_n)$ forms the bottom-up input (input layer F_0) to the field (layer) F_1 of n nodes whose activity vector is denoted by $\boldsymbol{X} = (X_1, ..., X_n)$. Each of the n nodes in field (layer) F_2 represents a class or category of inputs around a prototype (cluster seed, recognition category) generated during self-organising activity of ART 1. Adaptive pathways lead from each F_1 node to all F_2 nodes (bottom up

adaptive filter), and from each F_2 node to all F_1 nodes (top down adaptive filter). All paths are excitatory unless marked with a minus sign.

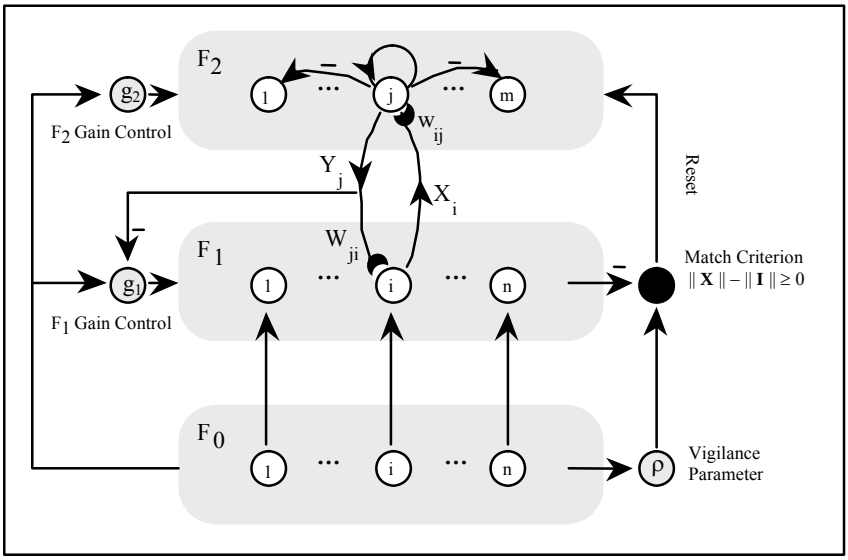


Figure 1 The ART 1 architecture

Carpenter and Grossberg designed the ART 1 network using previously developed building blocks based on biologically reasonable assumptions. The selection of a winner F_2 node, the top down and bottom up weight changes, and the enable/disable (reset) mechanism can all be described by realisable circuits governed by differential equations. The description of the ART 1 simulation algorithm below is adapted from Carpenter et al. (1991a, b). We consider the case where the competitive layer F_2 makes a choice and where the ART system is operating in a fast learning mode.

2.1 F_1-Activation

Each F_1 node can receive input from three sources: the $F_0 \rightarrow F_1$ bottom-up input; non-specific gain control signals; and top-down signals from the *m* nodes (winner-take-all units) of F_2, via an $F_2 \rightarrow F_1$ adaptive filter. A node is said to be *active* if it generates an output signal equal to 1. Output from inactive nodes equals 0. In ART 1 a F_1 node is active if at least two of the three input signals are large. This *rule for* F_1 *activation* is called the 2/3 *Rule* and realised in its simplest form as follows: The *i*th F_1 node is active if its net input exceeds a fixed threshold:

$$X_i = \begin{cases} 1 & \text{if } I_i + g_1 + \sum_{j=1}^{n} Y_j \, W_{ji} > 1 + \bar{W} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where I_i is the binary $F_0 \rightarrow F_1$ input, g_1 the binary non-specific F_1 gain control signal, and term $\Sigma Y_j W_{ji}$ the sum of $F_2 \rightarrow F_1$ signals Y_j via pathways with adaptive weights W_{ji}, and $\bar{W}$ $(0 < \bar{W} < 1)$ is a constant. Hereby the F_1 gain control g_1 is defined as

$$g_i = \begin{cases} 1 & \text{if } F_0 \text{ is active and } F_2 \text{ is inactive} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

It is important to note that F_2 activity inhibits g_1, as shown in Figure 1. These laws for F_1 activation imply that, if F_2 is inactive, then

$$X_i = \begin{cases} 1 & \text{if } I_i = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

If exactly one F_2 node J is active, the sum $\Sigma X_i W_{ji}$ in Equation (1) reduces to the single term W_{Ji}, so that

$$X_i = \begin{cases} 1 & \text{if } I_i = 1 \text{ and } W_{Ji} > \bar{W} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

2.2 Rules for Category Choice (F_2 choice)

F_2 nodes interact with each other by lateral inhibition. The result is a competitive winner-take all response. The set of committed F_2 nodes (prototypes) is defined as follows. Let T_j denote the total input from F_1 to the jth F_2 processor, given by

$$T_j = \sum_{i=1}^{n} X_i \, w_{ij}, \tag{5}$$

where w_{ij} represent the $F_1 \rightarrow F_2$ (i.e. bottom-up or forward) adaptive weights. If some $T_j > 0$, define the F_2 choice index J by

$$T_J = \max_{j=1,\ldots,m} \{T_j\}. \tag{6}$$

Characteristically, J is uniquely defined. Then the components of the F_2 output vector $\mathbf{Y} = (Y_1, \ldots, Y_m)$ are

$$Y_i = \begin{cases} 1 & \text{if } j = J \\ 0 & \text{if } j \neq J. \end{cases} \tag{7}$$

If two or more indices j share maximal input, then one of these is chosen at random.

2.3 Learning Laws: Top Down and Bottom Up Learning

The *learning laws* as well as the rules for choice and search, may be described, using the following notation. Let $\boldsymbol{A} = (A_1, ..., A_m)$ be a binary m-dimensional vector, then the norm of $\boldsymbol{A}$ is defined by

$$||\boldsymbol{A}|| = \sum_{i=1}^{m} |A_i| . \tag{8}$$

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be binary m-dimensional vectors, then a third binary m-dimensional vector $\boldsymbol{A} \cap \boldsymbol{B}$ may be defined by

$$(\boldsymbol{A} \cap \boldsymbol{B}) = 1 \qquad \text{if and only if } A_i = 1 \text{ and } B_i = 1. \tag{9}$$

All *ART* 1 *learning* is gated by F_2 activity. That is, the bottom up (forward) and the top down (backward or feedback) adaptive weights w_{iJ} and W_{Ji} can change only when the Jth F_2 node is active. Both types of weights are functions of the F_1 vector $\boldsymbol{X}$. Stated as a differential equation, the *top-down* or *feedback learning rule* is

$$\frac{d}{dt} W_{ji} = Y_j \, (X_i - W_{ji}), \tag{10}$$

where learning by W_{ji} is gated by Y_j. When the Y_j gate opens (i.e., when $Y_j > 0$), then learning begins and W_{ji} is attracted to X_i:

$$W_{ji} \rightarrow X_i. \tag{11}$$

In vector terms: if $Y_j > 0$, then $\boldsymbol{W}_j = (W_{j1}, ..., W_{jn})$ approaches $\boldsymbol{X} = (X_1, ..., X_n)$. Such a learning rule is termed *outstar learning rule* (Grossberg 1969). Initially, all W_{ji} are maximal, i.e.

$$W_{ji}(0) = 1. \tag{12}$$

Thus (with fast learning where the adaptive weights fully converge to equilibrium values in response to each input pattern) the top-down (feedback) weight vector

$\boldsymbol{W}_J$ is a binary vector at the start and end of each input presentation. By (3), (4), (9), (11) and (12), the *binary* F_1 *activity* (output) vector is given by

$$\boldsymbol{X} = \begin{cases} \boldsymbol{I} & \text{if } F_2 \text{ is inactive} \\ \boldsymbol{I} \cap \boldsymbol{W}_J & \text{if the } j\text{th } F_2 \text{ node is active} \end{cases} \tag{13}$$

When F_2 node J is active, by (4) and (10) learning causes

$$\boldsymbol{W}_J \rightarrow \boldsymbol{I} \cap \boldsymbol{W}_J \text{ (old)}. \tag{14}$$

In this *learning update rule* $\boldsymbol{W}_J$ (old) denotes $\boldsymbol{W}_J$ at the start of the current input presentation. By (11) and (13), $\boldsymbol{X}$ remains constant during learning, even though $|\boldsymbol{W}_J|$ may decrease. The first time an F_2 node J becomes active it is said to be *uncommitted*. Then, by (12)-(14)

$$\boldsymbol{W}_J \rightarrow \boldsymbol{I}. \tag{15}$$

The bottom up or forward weights have a slightly more complicated learning rule which leads to a similar, but normalised result. The combination with F_2 nodes which undergo cooperative and competitive interactions is called *competitive learning*. Initially all F_2 nodes are uncommitted. Forward weights w_{ij} in $F_1 \rightarrow F_2$ paths initially satisfy

$$w_{ij}(0) = \alpha_j, \tag{16}$$

where the parameters α_j are ordered according to $\alpha_1 > \alpha_2 > ... > \alpha_n$ for any admissible $F_0 \rightarrow F_1$ input $\boldsymbol{I}$.

Like the top-down weight vector $\boldsymbol{W}_J$, the bottom-up $F_1 \rightarrow F_2$ weight vector $\boldsymbol{w}_J = (w_{1J}, ..., w_{iJ}, ..., w_{nJ})$ also becomes proportional to the F_1 output vector $\boldsymbol{X}$ when the F_2 node J is active. But in addition the forward weights are scaled inversely to $||\boldsymbol{X}||$, so that

$$w_{iJ} \rightarrow \frac{X_i}{\beta + ||\boldsymbol{X}||} \tag{17}$$

with $\beta > 0$ (the small number β is included to break ties). This $F_1 \rightarrow F_2$ learning law (Carpenter and Grossberg 1987a) realises a type of competition among the weights $\boldsymbol{w}_J$ adjacent to a given F_2 node J. By (13), (14) and (17), during learning

$$w_J \rightarrow \frac{\boldsymbol{I} \cap \boldsymbol{W}_J(\text{old})}{\beta + ||\boldsymbol{I} \cap \boldsymbol{W}_J(\text{old})||}. \tag{18}$$

(18) establishes the update rule for forward weights. The w_{ij} initial values are required to be sufficiently small so that an input $\boldsymbol{I}$ which perfectly matches a previously learned vector $\boldsymbol{w}_J$ will select the F_2 node J rather than an uncommitted node. This is accomplished by assuming that

$$0 < \alpha_j = w_{ij}(0) < \frac{1}{\beta + ||\boldsymbol{I}||} \tag{19}$$

for all $F_1 \rightarrow F_2$ inputs $\boldsymbol{I}$. When $\boldsymbol{I}$ is first presented, $\boldsymbol{X} = \boldsymbol{I}$, so by (5), (14), (16), and (18), the $F_1 \rightarrow F_2$ input vector $T = (T_1, ..., T_m)$ obeys

$$\boldsymbol{T}_j(\boldsymbol{I}) = \sum_{i=1}^{n} I_i \; w_{ij} = \begin{cases} ||\boldsymbol{I}||\; \alpha_j & \text{if } j \text{ is an uncommitted node} \\ \dfrac{||\boldsymbol{I} \cap \boldsymbol{W}_J||}{\beta + ||\boldsymbol{W}_J||} & \text{if } j \text{ is a committed node.} \end{cases} \tag{20}$$

(20) is termed the choice function in ART 1, where β is the choice parameter and $\beta \neq 0$. The limit $\beta \rightarrow 0$ is called *conservative limit*, because small β-values tend to minimise recoding during learning. If β is taken so small then – among committed F_2 nodes – T_j is determined by the size $||\boldsymbol{I} \cap \boldsymbol{W}_j||$ relative to $||\boldsymbol{W}_j||$. Additionally, α_j values are taken to be so small that an uncommitted F_2 node will generate the maximum T_j value in (20) only if $||\boldsymbol{I} \cap \boldsymbol{W}_j|| = 0$ for all committed nodes. Larger values of α_j and β bias the system toward earlier selection of uncommitted nodes when only poor matches are to be found among the committed nodes (for a more detailed discussion see Carpenter and Grossberg 1987a).

2.4 Rules for Search

It is important to note that ART 1 overcomes the stability - plasticity dilemma by accepting and adapting the prototype of a category (class) stored in F_2 only when the input pattern is *sufficiently similar* to it. In this case, the input pattern and the stored prototype are said to *resonate* (hence the term *resonance* theory). When an input pattern fails to match any existing prototype (node) in F_2, a new category is formed (as in Hartigan's, 1975, leader algorithm), with the input pattern as the prototype, using a previously uncommitted F_2 unit. If there are no such uncommitted nodes left, then a novel input pattern gives no response (see Hertz et al. 1991).

A dimensionless parameter ρ with $0 < \rho \leq 1$ which is termed *vigilance parameter* establishes a matching (similarity) criterion for deciding whether the similarity is good enough for the input pattern to be accepted as an example of the chosen prototype. The degree of match (similarity) between bottom-up input $\boldsymbol{I}$ and top-down expectation $\boldsymbol{W}_j$ is evaluated at the orienting subsystem of ART 1 (see Figure 1) which measures whether prototype J adequately represents input pattern $\boldsymbol{I}$.

A *reset* occurs when the match fails to meet the criterion established by the parameter ρ.

In fast-learning ART 1 with choice at F_2, the *search process* may be characterised by the following steps:

Step 1: Select one F_2 node J that maximises T_j in (20), and read-out its top-down (feedback) weight vector $\boldsymbol{W}_J$.

Step 2: With J active, compare the F_1 output vector $\boldsymbol{X} = \boldsymbol{I} \cap \boldsymbol{W}_J$ with the $F_0 \rightarrow F_1$ input vector $\boldsymbol{I}$ at the orienting subsystem (see Figure 1).

Step 3A: Suppose that $\boldsymbol{I} \cap \boldsymbol{W}_J$ fails to match $\boldsymbol{I}$ at the level required by the ρ-criterion, i.e. that

$$||\boldsymbol{X}|| = ||\boldsymbol{I} \cap \boldsymbol{W}_J|| < \rho\, ||\boldsymbol{I}||. \tag{21}$$

This mismatch causes the system to reset and inhibits the winning node J for the duration of the input interval during which $\boldsymbol{I}$ remains on. The index of the chosen prototype (F_2 node) is reset to the value corresponding to the next highest $F_1 \rightarrow F_2$ input $\boldsymbol{T}_j$. With the new node active, Steps 2 and 3A are repeated until the chosen prototype satisfies the similarity [resonance] criterion (21).

Step 3B: Suppose that $\boldsymbol{I} \cap \boldsymbol{W}_J$ meets the similarity (match function) criterion, i.e.

$$||\boldsymbol{X}|| = ||\boldsymbol{I} \cap \boldsymbol{W}_J|| \geq \rho\, ||\boldsymbol{I}||, \tag{22}$$

then ART 1 search ends and the last chosen F_2 node J remains active until input $\boldsymbol{I}$ shuts off (or until ρ increases).

In this state, called *resonance*, both the feedforward ($F_1 \rightarrow F_2$) and the feedback ($F_2 \rightarrow F_1$) adaptive weights are updated if $\boldsymbol{I} \cap \boldsymbol{W}_J(\text{old}) \neq \boldsymbol{W}_J(\text{old})$. If ρ is chosen to be large (i.e. close to 1), the similarity condition becomes very stringent so that many finely divided categories (classes) are formed. A ρ–value close to zero gives a coarse categorisation. The vigilance level can be changed during learning.

Finally, it is worth noting that ART 1 is exposed to discrete presentation intervals during which an input is constant and after which F_1 and F_2 activities are set to zero. Discrete presentation intervals are implemented by means of the F_1 and F_2 gain control signals (g_1, g_2). Gain signal g_2 is assumed (like g_1 in (2)) to be 0 if F_0 is inactive. When F_0 becomes active, g_2 and F_2 signal thresholds are assumed to lie in a range where the F_2 node which receives the largest input signal can become active.

3 The ARTMAP Neural Network Architecture

ARTMAP is a neural network architecture designed to solve supervised pattern recognition problems. The architecture is called ARTMAP because it maps input vectors in $\Re^n$ (such as feature vectors denoting spectral values of a pixel) to output vectors in $\Re^m$ (with $m < n$), representing predictions such as land use categories, where mapping is learned by example from pairs $\{\boldsymbol{A}^{(p)}, \boldsymbol{B}^{(p)}\}$ of sequentially presented input and output vectors $p = 1, 2, 3, \ldots$ and $\boldsymbol{B}^{(p)}$ is the correct prediction given $\boldsymbol{A}^{(p)}$. Figure 2 illustrates the main components of a binary ARTMAP system. The system incorporates two ART 1 modules, ART_a and ART_b. Indices *a* and *b* identify terms in the ART_a and ART_b modules, respectively. Thus, for example ρ_a and ρ_b denote the ART_a and ART_b vigilance (similarity) parameters, respectively.

During supervised learning ART_a and ART_b read vector inputs $\boldsymbol{A}$ and $\boldsymbol{B}$. The ART_a complementing coding preprocessor transforms the vector $\boldsymbol{A} = (A_1, \ldots, A_{na})$ into the vector $\boldsymbol{I}_a = (\boldsymbol{A}, \boldsymbol{A}^C)$ at the ART_a field F_0^a, where $\boldsymbol{A}^C$ denotes the complement of $\boldsymbol{A}$. The complement coded input $\boldsymbol{I}_a$ to the recognition system is the 2_{na}-dimensionable vector

$$\boldsymbol{I}_a = (\boldsymbol{A}, \boldsymbol{A}^C) = (A_1, \ldots, A_{na}, A_1^C, \ldots, A_{na}^C), \tag{23}$$

where

$$A_i^C = 1 - A_i . \tag{24}$$

Complement coding achieves normalisation while preserving amplitude information (see Carpenter et al. 1991a). $\boldsymbol{I}_a$ is the input to the ART field F_1^a. Similarly, the input to the ART_b field F_1^b is the vector $\boldsymbol{I}_b = (\boldsymbol{B}, \boldsymbol{B}^C)$.

If ART_a and ART_b were disconnected, each module would self-organise category groupings for the separate input sets $\{\boldsymbol{A}^{(p)}\}$ and $\{\boldsymbol{B}^{(p)}\}$, respectively, as described in Section 2. In an ARTMAP architecture design, however, ART_a and ART_b are connected by an inter-ART module, including a map field that controls the learning of an associative map from ART_a recognition categories (i.e. compressed representations of classes of examplars $\boldsymbol{A}^{(p)}$) to ART_b recognition categories (i.e. compressed representations of classes of examplars $\boldsymbol{B}^{(p)}$). Because the map field is the interface, where signals from F_2^a and F_2^b interact, it is denoted by F^{ab}. The nodes of F^{ab} have the same index j ($j = 1, \ldots, m_b$) as the nodes of F_2^b because there is a one-to-one correspondence between these sets of nodes.

ART_a and ART_b operate as outlined in Section 2 with the following additions. First, the ART_a vigilance (similarity) parameter ρ_a can increase during inter-ART reset according to the match tracking rule. Second, the map field F^{ab} can prime ART_b. This means, if F^{ab} sends non-uniform input to F_2^b in the absence of an $F_0^b \rightarrow F_1^b$ input $\boldsymbol{B}$, then F_2^b remains inactive. But as soon as an input arrives, F_2^b selects the node J receiving the largest $F^{ab} \rightarrow F_2^b$ input. Node J, in turn,

sends to F_1^b the top-down input weight vector W_j^b. Rules for the control strategy, called match tracking, are specified in the sequel (Carpenter et al. 1991a).

Let $\boldsymbol{X}^a = (X_1^a, ..., X_{na}^a)$ denote the F_1^a output vector and $\boldsymbol{Y}^a = (Y_1^a, ..., Y_{ma}^a)$ the F_2^a output vector. Similarly, let denote $\boldsymbol{X}^b = (X_1^b, ..., X_{nb}^b)$ the F_1^b output vector and $\boldsymbol{Y}^b = (Y_1^b, ..., Y_{mb}^b)$ the F_2^b output vector. The map field F^{ab} has m_b nodes and binary output vector $\boldsymbol{X}^{ab}$. Vectors $\boldsymbol{X}^a$, $\boldsymbol{Y}^a$, $\boldsymbol{X}^b$, $\boldsymbol{Y}^b$ and $\boldsymbol{X}^{ab}$ are set to the zero vector, **0**, between input presentations.

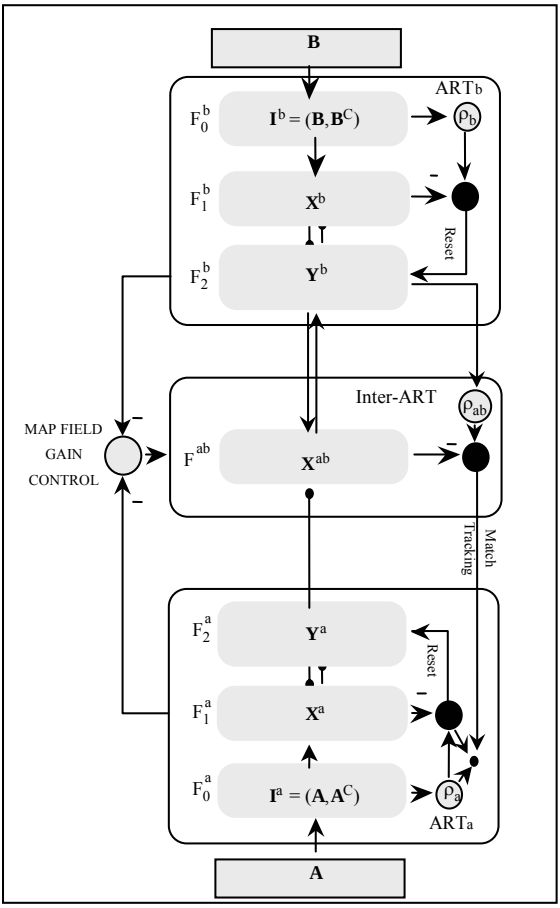


Figure 2 Block diagram of an ARTMAP system

The $F_2^a \rightarrow F^{ab}$ adaptive weights z_{kj} with $k = 1, ..., m_a$ and $j = 1, ..., m_b$ obey an outstar learning law similar to that governing the $F_2^b \rightarrow F_1^b$ weights, namely

$$\frac{d}{dt} z_{kj} = \boldsymbol{Y}_k^a (\boldsymbol{X}_k^{ab} - z_{kj}). \tag{25}$$

Each vector $(z_{k1}, ..., z_{kmb})$ is denoted by z_k. According to the learning rule established by (25), the $F_2^a \to F^{ab}$ weight vector $\mathbf{z}_k$ approaches the map field F^{ab} activity vector X^{ab} if the *k*th F_2^a node is active. Otherwise $\mathbf{z}_k$ remains constant. If node *k* has not yet learned to make a prediction, all weights z_{ki} are set equal to 1, using an assumption, analogous to Equation (12), i.e. $z_{ki}(0) = 1$ for $k = 1, ..., m_a$, and $j = 1, ..., m_b$.

During resonance with ART_a category *k* active, $\mathbf{z}_K \to \boldsymbol{X}^{ab}$. In fast learning, once *k* learns to predict the ART_b category *J*, that association is permanent (i.e. $z_{KJ} = 1$ and $z_{Ki} = 0$ with $j \neq J$ for all time). The F^{ab} output vector $\boldsymbol{X}^{ab}$ obeys

$$\boldsymbol{X}^{ab} = \begin{cases} \boldsymbol{Y}^b \cap \mathbf{z}_K & \text{if the } k\text{th } F_2^a \text{ node is active and } F_2^b \text{ is active} \\ \mathbf{z}_K & \text{if } \textit{the } k\text{th } F_2^a \text{ node is active and } F_2^b \text{ is inactive} \\ \boldsymbol{Y}^b & \text{if } F_2^a \text{ is inactive and } F_2^b \text{ is active} \\ 0 & \text{if } F_2^a \text{ is inactive and } F_2^b \text{ is inactive.} \end{cases} \tag{26}$$

When ART_a makes a prediction that is incompatible with the actual ART_b input (i.e. z_k is disconfirmed by $\boldsymbol{Y}^b$), then this mismatch triggers on ART_a search for a new category as follows. At the start of each input presentation the ART_a vigilance (similarity) parameter ρ_a equals a baseline vigilance ρ_a. The map field vigilance parameter is ρ_{ab}. If a mismatch at F^{ab} occurs, i.e. if

$$||\boldsymbol{X}^{ab}|| < \rho_a ||\boldsymbol{I}^b||, \tag{27}$$

then match tracking is triggered to search a new F_2^a node. Match tracking starts a cycle of ρ_a adjustment and increases ρ_a until it is slightly higher than the F_1^a match value $||\boldsymbol{A} \cap \boldsymbol{W}_k^a|| \, ||\boldsymbol{I}^a||^{-1}$, where $\boldsymbol{W}_k^a$ denotes the top-down $F_2^a \to F_1^a$ ART_a weight vector $(\boldsymbol{W}_1^a, ..., \boldsymbol{W}_{na}^a)$. Then

$$||\boldsymbol{X}^a|| = ||\boldsymbol{I}^a \cap \boldsymbol{W}_k^a|| < \rho_a ||\boldsymbol{I}^a|| \tag{28}$$

where $\boldsymbol{I}^a$ is the current ART_a input vector and *k* is the index of the active F_2^a node. When this occurs, ART_a search leads either to ARTMAP resonance, where a newly chosen F_2^a node *K* satisfies both the ART_a matching criterion (see also Equation (21)):

$$||\boldsymbol{X}^a|| = ||\boldsymbol{I}^a \cap \boldsymbol{W}_K^a|| \geq \rho_a ||\boldsymbol{I}^a|| \tag{29}$$

and the map field matching criterion:

$$||\boldsymbol{X}^{ab}|| = ||\boldsymbol{I}^b \cap \mathbf{z}_K|| \geq \rho_{ab} ||\boldsymbol{Y}^b|| \tag{30}$$

or, if no such node K exists, to the shut-down of F_2^a for the remainder of the input presentation (Carpenter et al. 1993).

4 Generalisation to Fuzzy ARTMAP

Fuzzy ARTMAP has been proposed by Carpenter et al. (1991b) as a direct generalisation of ARTMAP for supervised learning of recognition categories and multidimensional maps in response to arbitrary sequences of continuous-valued (and binary) patterns not necessarily interpreted as fuzzy set of features. The generalisation to learning continuous and binary input patterns is achieved by using fuzzy set operations rather than standard binary set theory operations (see Zadeh 1965). Figure 3 summarises how the crisp logical ARTMAP operations of category choice, matching and learning translate into fuzzy ART operations when the crisp (non-fuzzy or hard) intersection operator ($\cap$) of ARTMAP is replaced by the fuzzy intersection or (component-wise) minimum operator ($\wedge$). Due to the close formal homology between ARTMAP and fuzzy ARTMAP operations (as illustrated in Figure 3), there is no need to describe fuzzy ARTMAP in detail here, but for a better understanding it is important to stress differences to the ARTMAP approach.

Fuzzy ARTMAP in its most general form inherits the architecture as outlined in Figure 2 and employs two fuzzy ART modules as substitutes for the ART 1 subsystems. It is noteworthy that fuzzy ART reduces to ART 1 in response to binary input vectors (Carpenter et al. 1993). Associated with each F_2^a [F_2^b] node $k = 1, ..., m_a$ [$j = 1, ..., m_b$] is a vector $\boldsymbol{W}_k^a[\boldsymbol{W}_j^b]$ of adaptive weights which subsumes both the bottom-up and top-down weight vectors of ART 1.

Fuzzy ARTMAP dynamics are determined by a choice parameter $\beta > 0$, a learning parameter $\gamma \in [0,1]$; and three vigilance (similarity) parameters: the ART_a vigilance parameter ρ_a, the ART_b vigilance parameter ρ_b and the map field vigilance parameter ρ_{ab} with $\rho_a, \rho_b, \rho_{ab} \in]0,1]$. The choice functions $T_k(\boldsymbol{A})$ and $T_j(\boldsymbol{B})$ are defined as in Figure 3, where the fuzzy intersection ($\wedge$) for any n-dimensional vectors $\boldsymbol{S} = (S_1, ..., S_n)$ and $\boldsymbol{T} = (T_1, ..., T_n)$ is defined by

$$(\boldsymbol{S} \wedge \boldsymbol{T})_l = \min_l (S_l, T_l) \tag{31}$$

The fuzzy choice functions $T_k(\boldsymbol{A})$ and $T_j(\boldsymbol{B})$ (see Figure 3) can be interpreted as a fuzzy membership of the input $\boldsymbol{A}$ in the kth category and the input $\boldsymbol{B}$ in the jth category, respectively. In the conservative limit (i.e. $\beta \to 0$) the choice function $T_k(\boldsymbol{A})$ primarily reflects the degree to which the weight vector $\boldsymbol{W}_k^a$ is a fuzzy subset of the input vector $\boldsymbol{A}$. If

$$\frac{||\boldsymbol{I}_l^a \wedge \boldsymbol{W}_k^a||}{||\boldsymbol{W}_k^a||} = 1, \tag{32}$$

	Binary ARTMAP	*Fuzzy ARTMAP*
ART_a *category choice* [*choice parameter*]	$T_k(\boldsymbol{I}^a) = \dfrac{\|\boldsymbol{I}^a \cap \boldsymbol{W}_k^a\|}{+\|\boldsymbol{W}_k^a\|}$	$T_k(\boldsymbol{I}^a) = \dfrac{\|\boldsymbol{I}^a \wedge \boldsymbol{W}_k^a\|}{+\|\boldsymbol{W}_k^a\|}$
ART_b *category choice* [*choice parameter*]	$T_j(\boldsymbol{I}^b) = \dfrac{\|\boldsymbol{I}^b \cap \boldsymbol{W}_j^b\|}{+\|\boldsymbol{W}_j^b\|}$	$T_j(\boldsymbol{I}^b) = \dfrac{\|\boldsymbol{I}^b \wedge \boldsymbol{W}_j^b\|}{+\|\boldsymbol{W}_j^b\|}$
ART_a *matching criterion* [ρ_a ART_a *vigilance parameter*]	$\|\boldsymbol{I}^a \cap \boldsymbol{W}_k^a\| \geq \rho_a \|\boldsymbol{A}\|$	$\|\boldsymbol{I}^a \wedge \boldsymbol{W}_k^a\| \geq \rho_a \|\boldsymbol{A}\|$
ART_b *matching criterion* [ρ_b ART_b *vigilanc parameter*]	$\|\boldsymbol{I}^b \cap \boldsymbol{W}_j^b\| \geq \rho_b \|\boldsymbol{B}\|$	$\|\boldsymbol{I}^a \wedge \boldsymbol{W}_j^b\| \geq \rho_b \|\boldsymbol{B}\|$
Map field F^{ab} *matching criterion* [ρ_{ab} *Map field vigilance parameter*]	$\|\boldsymbol{X}^{ab}\| = \|\boldsymbol{Y}^b \cap \mathbf{z}_k\| \geq \rho_{ab} \|\boldsymbol{Y}^b\|$	$\|\boldsymbol{X}^{ab}\| = \|\boldsymbol{Y}^b \wedge \mathbf{z}_k\| \geq \rho_{ab} \|\boldsymbol{Y}^b\|$
ART_a $F_2^a \rightarrow F_1^a$ *Learning weight updates* [γ *learning parameter*]	$\boldsymbol{W}_k^a(\text{new}) = \gamma(\boldsymbol{A} \cap \boldsymbol{W}_k^a(\text{old})) + (1-\gamma)\ \boldsymbol{W}_k^a(\text{old})$	$\boldsymbol{W}_k^a(\text{new}) = \gamma(\boldsymbol{A} \wedge \boldsymbol{W}_k^a(\text{old})) + (1-\gamma)\ \boldsymbol{W}_k^a(\text{old})$
ART_b $F_2^b \rightarrow F_1^b$ *Learning weight updates* [γ *learning parameter*]	$\boldsymbol{W}_j^b(\text{new}) = \gamma(\boldsymbol{B} \cap \boldsymbol{W}_j^b(\text{old})) + (1-\gamma)\ \boldsymbol{W}_j^b(\text{old})$	$\boldsymbol{W}_j^b(\text{new}) = \gamma(\boldsymbol{B} \wedge \boldsymbol{W}_j^b(\text{old})) + (1-\gamma)\ \boldsymbol{W}_j^b(\text{old})$

Figure 3 Comparison between binary and fuzzy ARTMAP [$\cap$ denotes the crisp AND (intersection) operator and $\wedge$ its fuzzy counterpart]

then $\boldsymbol{W}_k^a$ is a fuzzy subset of $\boldsymbol{I}^a$ and category k is said to be a fuzzy subset choice for input $\boldsymbol{I}^a$. When a fuzzy subset exists, it is always selected over other choices.

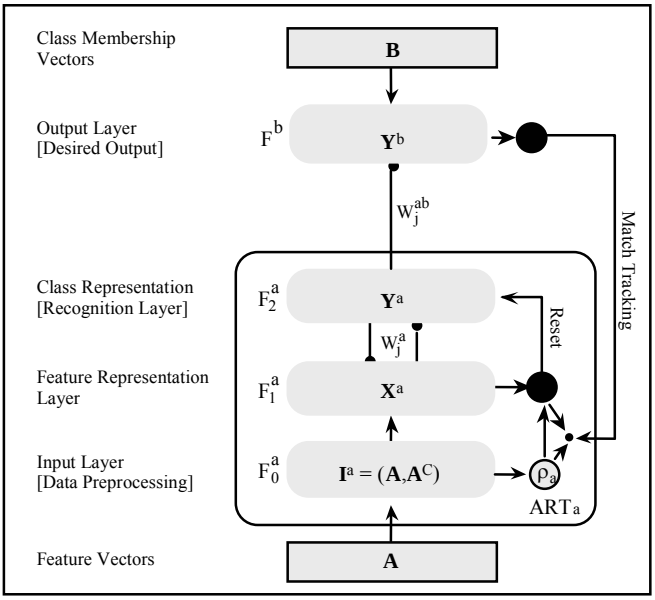


Figure 4 The fuzzy ARTMAP classifier: A simplified ARTMAP architecture

The same holds true for $T_j(\boldsymbol{I}^b)$. (Carpenter et al. 1992). Resonance depends on the degree to which $\boldsymbol{I}^a$ [$\boldsymbol{I}^b$] is a fuzzy set of $\boldsymbol{W}_k^a$ [$\boldsymbol{W}_k^b$], by the matching criteria (or functions) outlined in Figure 3. The close linkage between fuzzy subsethood and ART choice, matching and learning forms the foundations of the computational features of fuzzy ARTMAP (Carpenter et al. 1992). Especially if category K is a fuzzy subset ART_a choice, then the ART_a match function value ρ_a is given by

$$\rho_a = \frac{||\boldsymbol{I}^a \wedge \boldsymbol{W}_K^a||}{||\boldsymbol{I}^a||} = \frac{||\boldsymbol{W}_K^a||}{||\boldsymbol{I}^a||}. \tag{33}$$

Once search ends, the ART_a weight vector $\boldsymbol{W}_K^a$ is updated according to the equation

$$\boldsymbol{W}_K^a(\text{new}) = \gamma(\boldsymbol{A} \wedge \boldsymbol{W}_K^a(\text{old})) + (1-\gamma)\boldsymbol{W}_K^a(\text{old}) \tag{34}$$

and similarly the ART_b weight vector $\boldsymbol{W}_J^b$:

$$\boldsymbol{W}_J^b(\text{new}) = \gamma(\boldsymbol{B} \wedge \boldsymbol{W}_J^b(\text{old})) + (1-\gamma)\boldsymbol{W}_J^b(\text{old}) \tag{35}$$

where $\gamma = 1$ corresponds to fast learning as described in Figure 3.

The aim of fuzzy ARTMAP is to correctly associate continuous valued ART_a inputs with continuous valued ART_b inputs. This is accomplished indirectly by associating categories formed in ART_a with categories formed in ART_b. For a pattern classification problem at hand, the desired association is between a continuous valued input vector and some categorical code which takes on a discrete set of values representing the a priori given classes. In this situation the ART_b network is not needed because the internal categorical representation which ART_b would learn already exists explicitly. Thus, the ART_b and the map field F^{ab} can be replaced by a single F^b as shown in Figure 4.

5 The Spectral Pattern Recognition Problem

The spectral pattern recognition problem considered here is the supervised pixel-by-pixel classification problem in which the classifier is trained with examples of the classes (categories) to be recognised in the data set. This is achieved by using limited ground survey information which specifies where examples of specific categories are to be found in the imagery. Such ground truth information has been gathered on sites which are well representative of the much larger area analysed from space. The image data set consists of 2,460 pixels (resolution cells) selected from a Landsat-5 Thematic Mapper (TM) scene (270 x 360 pixels) from the city of Vienna and its northern surroundings (observation date: June 5, 1985; location of the center: 16˚23' E, 48˚14' N; TM Quarter Scene 190-026/4). The six Landsat TM spectral bands used are blue (SB1), green (SB2), red (SB3), near IR (SB4), mid IR (SB5) and mid IR (SB7), excluding the thermal band with only a 120 m ground resolution. Thus, each TM pixel represents a ground area of 30 x 30 m^2 and has six spectral band values ranging over 256 digital numbers (8 bits).

The purpose of the multispectral classification task at hand is to distinguish between the eight categories of urban land use listed in Table 1. The categories chosen are meaningful to photo-interpreters and land use managers, but are not necessarily spectrally homogeneous. This prediction problem, used to evaluate the performance of fuzzy ARTMAP in a real world context, is challenging. The pixel-based remotely sensed spectral band values are noisy and sometimes unreliable. The number of training sites is small relative to the number of land use categories (one-site training case). Some of the urban land use classes are sparsely distributed in the image. Conventional statistical classifiers such as the Gaussian maximum likelihood classifier have been reported to fail to discriminate spectrally inhomogeneous classes such as C6 (see, e.g. Hepner et al. 1990). Thus, there is evidently a need for new more powerful tools (Barnsley 1993).

Table 1 Categories used for classification and number of training/testing pixels

Category	*Description of the category*	*Pixels*	
		Training	*Testing*
*C*1	Mixed grass and arable farmland	167	83
*C*2	Vineyards and areas with low vegetation cover	285	142
*C*3	Asphalt and concrete surfaces	128	64
*C*4	Woodland and public gardens with trees	402	200
*C*5	Low density residential and industrial areas (suburban)	102	52
*C*6	Densely built up residential areas (urban)	296	148
*C*7	Water courses	153	77
*C*8	Stagnant water bodies	107	54
Total number of pixels for training and testing		1,640	820

Ideally, the ground truth at every pixel of the scene should be known. Since this is impractical, one training site was chosen for each of the eight above mentioned land use categories. The training sites vary between 154 pixels (category: suburban) and 602 pixels (category: woodland and public gardens with trees). The above mentioned six TM bands provide the data set input for each pixel, with values scaled to the interval [0,1]. This approach resulted in a data base consisting of 2,460 pixels (about 2.5 percent of all the pixels in the scene) that are described by six-dimensional feature vectors, each tagged with its correct category membership. The set was divided into a training set (two thirds of the training site pixels) and a testing set by stratified random sampling, stratified in terms of the eight categories. Pixels from the testing set are not used during network training (parameter estimation) and serve only to evaluate out-of-sample test (prediction, generalisation) performance accuracy when the trained classifier is presented with novel data. The goal is to predict the correct land use category for the test sample of pixels.

A good classifier is one which after training with the training set of pixels is able to predict pixel assignments over much wider areas of territory from the remotely sensed data without the need for further ground survey (see Wilkinson et al. 1995). The performance of any classifier is, thus, dependent upon three factors: the adequacy of the training set of pixels and, therefore, the choice of the training sites; the in-sample performance of the classifier; and the out-of-sample or generalisation performance of the trained classifier. Of these three factors the first is often outside the control of the data analyst, and thus outside of the scope of this chapter.

6 Fuzzy ARTMAP Simulations and Classification Results

In this real world setting, fuzzy ARTMAP performance is examined and compared with that of the multi-layer perceptron and that of the conventional maximum likelihood classifier. (In-sample and out-of-sample) Performance is measured in

terms of the fraction of the total number of correctly classified pixels (i. e. the sum of the elements along the main diagonal of the classification error matrix).

During training and testing, a given pixel provides an ART_a input $\boldsymbol{A} = (A_1, A_2, A_3, A_4, A_5, A_6)$ where A_1 is the blue, A_2 the green, A_3 is the red, A_4 the near infrared, A_5 and A_6 the mid infrared (1.55-1.75 μm and 2.08-2.35 μm, respectively) spectral band values measured at each pixel. The corresponding ART_b input vector $\boldsymbol{B}$ represents the correct land use category of the pixel's site:

$$\boldsymbol{B} = \begin{cases} (1,0,0,0,0,0,0,0) & \text{for mixed grass and arable farmland; category 1} \\ (0,1,0,0,0,0,0,0) & \text{for vineyards and areas with low vegetation cover, category 2} \\ (0,0,1,0,0,0,0,0) & \text{for asphalt and concrete surfaces; category 3} \\ (0,0,0,1,0,0,0,0) & \text{for woodland and public gardens with trees; category 4} \\ (0,0,0,0,1,0,0,0) & \text{for low density residential and industrial areas; category 5} \\ (0,0,0,0,0,1,0,0) & \text{for densely built-up residential areas; category 6} \\ (0,0,0,0,0,0,1,0) & \text{for water courses; category 7} \\ (0,0,0,0,0,0,0,1) & \text{for stagnant water bodies; category 8.} \end{cases}$$

During training vector $\boldsymbol{B}$ informs the fuzzy ARTMAP classifier of the land use category to which the pixel belongs. This supervised learning process allows adaptive weights to encode the correct associations between $\boldsymbol{A}$ and $\boldsymbol{B}$. The remote sensing problem described in Section 5 requires a trained fuzzy ARTMAP network to predict the land use category of the test set pixels, given six spectral band values measured at each pixel.

Following a search, if necessary, the classifier selects an ART_a category by activating an F_a^2 node K for the chosen pixel, and learns to associate category K with the ART_b land use category of the pixel. With fast learning ($\gamma = 1$), the class prediction of each ART_a category K is permanent. If some input $\boldsymbol{A}$ with a different class prediction later chooses this category, match tracking will raise vigilance p_a just enough to trigger a search for a different ART_a category. If the finite input set is presented repeatedly, then all training set inputs learn to predict with 100 % classification accuracy, but start to fit noise present in the remotely sensed spectral band values.

Fuzzy ARTMAP is trained incrementally, with each spectral band vector $\boldsymbol{A}$ presented just once. Following a search, if necessary, the classifier selects an ART_a category by activating an F_2^a node K for the chosen pixel, and learns to associate category K with the ART_b land use category of the pixel. With fast learning ($\gamma = 1$), the class prediction of each ART_a category K is permanent. If some input $\boldsymbol{A}$ with a different class prediction later chooses this category, match tracking will raise vigilance ρ_a just enough to trigger a search for a different ART_b category. If the finite input set is presented repeatedly, then all training set inputs learn to predict with 100% classification accuracy, but start to fit noise present in the remotely sensed spectral band values.

All the simulations described below use the simplified fuzzy ARTMAP architecture outlined in Figure 3, with three parameters only: a choice parameter

$\beta > 0$, the learning parameter $\gamma = 1$ (fast learning), and an ART_a vigilance parameter $\rho_a \in [0,1]$. In each simulation, the training data set represents 1,640 pixels and the testing data set 820 pixels. Fuzzy ARTMAP was run with five different random orderings of the training and test sets, since input order may affect in-sample and out-of-sample performance. All simulations were carried out at the Department of Economic Geography (WU Wien) on a SunSPARCserver 10-GS with 128 MB RAM.

Table 2 Fuzzy ARTMAP simulations of the remote sensing classification problem: The effect of variations in choice parameter β ($\rho_a = 0.0$)

Choice parameter β	*Out-of-sample performance*	*Number of* F_a^2 *nodes*	*Number of epochs*
$\beta = 0.001$			
Run 1	98.54	125	6
Run 2	99.26	116	8
Run 3	99.51	121	6
Run 4	99.39	126	6
Run 5	99.75	148	7
Average	99.29	127	6.5
$\beta = 0.1$			
Run 1	99.26	126	7
Run 2	99.90	115	6
Run 3	99.36	115	7
Run 4	99.51	124	7
Run 5	99.26	127	7
Average	99.26	121	7
$\beta = 1.0$			
Run 1	99.98	218	10
Run 2	98.17	202	8
Run 3	98.90	212	8
Run 4	98.50	236	10
Run 5	98.40	232	10
Average	98.75	220	9

Table 2 summarises out-of-sample performance (measured in terms of classification accuracy) on 15 simulations, along with the number of ART_a categories generated and the number of epochs needed to reach any asymptotic training set performance (i. e. about 100% in-sample classification accuracy; Figure 5 shows how in-sample and out-of-sample performance changes depending on the number of training epochs of fuzzy ARTMAP). Each run had a different, randomly chosen presentation order for the 1,640 training and the 820 testing vectors. The choice parameter β was

set, first, near the conservative limit at value $\beta = 0.001$, and then at the higher values of $\beta = 0.1$ and $\beta = 1.0$. These β–value inputs were repeatedly presented in a given random order until 100% training classification accuracy was reached. This required six to eight epochs in the cases of $\beta = 0.001$ and $\beta = 0.1$, while for $\beta = 1.0$ eight to ten epochs were necessary. There seems to be a tendency that the number of epochs needed for 100% training set performance is increasing with higher β–values. All simulations used fast learning ($\gamma = 1.0$), which generates a distinct ART_a category structure for each input ordering. The number of F_2^a nodes ranged from 116 to 148 in the case of $\beta = 0.001$, 115 to 127 in the case of $\beta = 0.1$, and 202 to 236 in the case of $\beta = 1.0$. This tendency of increasing number of ART_a categories with increasing β–values and increasing training time is illustrated in Figure 6. All simulations used $\rho_a = 0.0$ which tends to minimise the number of F_2^a nodes compared with higher ρ_a-values not shown in Table 2. The best average result (averaged over five independent simulation runs) was obtained with $\beta = 0.01$ and 6.5 epoch training (99.29 % classification accuracy). All the 15 individual simulation runs reached an out-of-sample performance close to 100 % (range: 98.40 to 99.90 %).

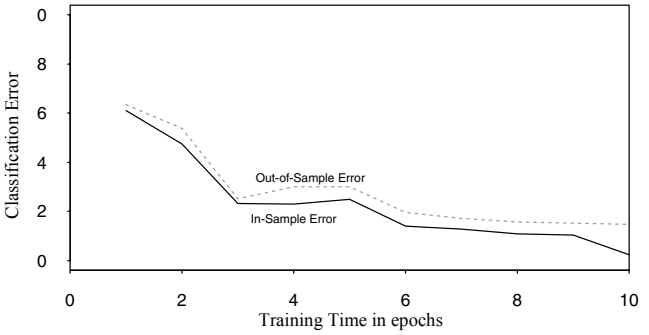


Figure 5 In-sample and out-of-sample classification error during training ($\beta = 0.001$, $\gamma = 1.0$, $\rho_a = 0.001$)

Table 3 shows how in-sample and out-of-sample performance changes depending on the number of F_2^a nodes with $\rho_a = 0.95$, 0.75, 0.50 and 0.0. In these simulations, learning is incremental, with each input presented only once (in ART terminology: one epoch training). The choice parameter is set to $\beta = 0.001$. The best overall results, in terms of average in-sample and out-of-sample performance were obtained with an ART_a vigilance close to one (96.36 % and 95.82 %, respectively). For $\rho_a = 0.0$ the in-sample and out-of-sample performances decline to 91.69 % and 91.06 %, respectively. But the runs with $\rho_a = 0.0$ use much fewer ART_a categories (32 to 44) compared to $\rho_a = 0.95$ (276 to 298 ART_a categories), and generate stable performance over the five runs. Increasing vigilance creates more ART_a categories. One final note to be made here is that most fuzzy ARTMAP learning occurs on the first epoch, with the test set performance on

systems trained for one epoch typically over 92 % that of systems exposed to inputs for six to eight epochs (compare Table 3 with Table 2).

Table 3 Fuzzy ARTMAP simulations of the remote sensing classification problem: The effect of variations in vigilance ρ_a ($\beta = 0.001$)

Vigilance (*similarity*) *parameter* ρ_a	*In-sample performance*	*Out-of-sample performance*	*Number of* F_2^a *nodes*
$\rho_a = 0.95$			
Run 1	97.01	96.20	285
Run 2	97.00	96.20	298
Run 3	96.15	95.60	276
Run 4	96.21	95.36	276
Run 5	95.06	94.39	286
Average	96.36	95.82	284
$\rho_a = 0.75$			
Run 1	93.00	92.00	52
Run 2	92.26	93.29	47
Run 3	91.82	90.00	42
Run 4	93.00	93.04	53
Run 5	90.31	91.83	53
Average	92.08	92.03	50
$\rho_a = 0.50$			
Run 1	92.20	91.40	43
Run 2	90.20	89.51	43
Run 3	94.45	94.76	44
Run 4	93.35	93.42	43
Run 5	92.98	93.90	45
Average	92.62	92.59	44
$\rho_a = 0.0$			
Run 1	90.70	90.60	35
Run 2	92.26	91.22	44
Run 3	90.97	90.30	34
Run 4	91.95	90,73	40
Run 5	92.56	92.44	32
Average	91.69	91.06	37

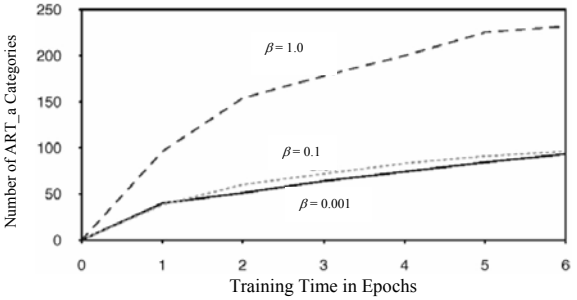


Figure 6 Effect of choice parameter β on the number (m_a) of ART_a categories (γ = 1.0, ρ_a = 0.001)

Table 4 summarises the results of the third set of fuzzy ARTMAP simulations carried out, in terms of both in-sample and out-of-sample performance along with the number of F_2^a nodes. The choice parameter β was set near the conservative limit at value β = 0.001 and ART_a vigilance at ρ_a = 0.0. Training lasted for one epoch only. As training size increases from 164 to 164,000 pixel vectors both in-sample and out-of-sample performances increase, but so does the number of ART_a category nodes. In-sample classification accuracy increases from 83.2 % to 99.3 %, and out-of-sample classification accuracy from 80.1 % to 99.2 %, while the number of ART_a category nodes increases from 19 to 225. Each category node K requires six learned weights $\boldsymbol{W}_K^a$ in ART_a. One epoch training on 164 training pixels creates 19 ART_a categories and so uses 72 ART_a adaptive weights to achieve 80.1 % out-of-sample classification accuracy (820 test pixels), while one epoch training on 164,000 pixels requires 225 ART_a categories and, thus, 1,350 ART_a adaptive weights to arrive at an out-of-sample performance of 99.2 %. Evidently, the fuzzy ARTMAP classifier becomes arbitrarily accurate provided the number of F_2^a nodes increases as needed.

Table 4 Fuzzy ARTMAP simulations of the remote sensing classification problem: The effect of variations in training size (ρ_a = 0.0, β = 0.001)

Number of training pixels	*In-sample performance*	*Out-of-sample performance*	*Number of F_2^a nodes*
164	83.2	80.1	19
1,640	93.0	92.0	33
16,400	99.3	99.2	135
164,000	99.3	99.2	225

Finally, fuzzy ARTMAP performance is compared with that of a multi-layer perceptron classifier as developed and implemented in Fischer et al. (1994), using the same training and testing set data. Table 5 summarises the results of the comparison of the two neural classifers in terms of the in-sample and out-of-sample classification accuracies along with the number of epochs (i. e. one pass through the

training data set) and the number of hidden units/ ART_a category nodes (a hidden unit is somewhat analogous to an ART_a category for purposes of comparison) to reach asymptotic convergence, and the number of adaptive weight parameters.

The fuzzy ARTMAP classifier has been designed with the following specifications: choice parameter near the conservative limit at value $\beta = 0.001$, learning parameter $\gamma = 1.0$, constant ART_a vigilance $\rho_a = 0.0$, repeated presentation of inputs in a given order until 100% training set performance was reached. Stability and match tracking allow fuzzy ARTMAP to construct automatically as many ART_a categories as are needed to learn any consistent training set to 100% classification accuracy. The multi-layer perceptron classifier is a pruned one hidden layer feedforward network with 14 logistic hidden units and eight softmax output units, using an epoch-based stochastic version of the backpropagation algorithm (epoch size: three training vectors, no momentum update, learning parameter $\gamma = 0.8$). The Gaussian maximum likelihood classifier based on parametric density estimation by maximum likelihood was chosen because it represents a widely used standard for comparison that yields minimum total classification error for Gaussian class distributions.

Table 5 Performance of fuzzy ARTMAP simulations of the remote sensing classification problem: Comparison with the multi-layer perceptron and the Gaussian maximum likelihood classifier

Classifier	*Epochs*[1]	*Hidden units/* ART_a *categories*	*Adaptive weight parameters*	*In-sample classification accuracy*	*Out-of-sample classification accuracy*
Fuzzy ARTMAP	8	116	812	100.00	99.26
Multi-Layer Perceptron	92	14	196	92.13	89.76
Gaussian ML	–	–	–	90.85	85.24

[1] one pass through the training data set

Fuzzy ARTMAP: $\beta = 0.001$, $\gamma = 1.0$, $\rho_a = 0.0$, asymptotic training

Multi-layer perceptron: logistic hidden unit activation, softmax output unit activation, network pruning, epoch-based stochastic version of backpropagation with epoch size of three, learning rate $\gamma = 0.8$

The fuzzy ARTMAP classifier has an outstanding out-of-sample classification accuracy of 99.26 % on the 820 pixels testing data set. Thus the error rate (0.74 %) is less than 1/15 that of the multi-layer perceptron and 1/20 that of the Gaussian maximum likelihood classifier. A more careful inspection of the classification error (confusion) matrices (see Appendix) shows that there is a some confusion between the urban (densely built-up residential) areas and water courses land use categories in the case of both the multi-layer perceptron and the Gaussian maximum likelihood classifiers, which is peculiar. The water body in this case is the river Danube that flows through the city and is surrounded by densely built up areas. The confusion could be caused by the 'boundary problem' where there are

mixed pixels at the boundary. The fuzzy ARTMAP neural network approach evidently accommodates more easily a heterogeneous class label such as *densely built-up residential areas* to produce a visually and numerically correct map, even with smaller numbers of training pixels (see Figure 7).

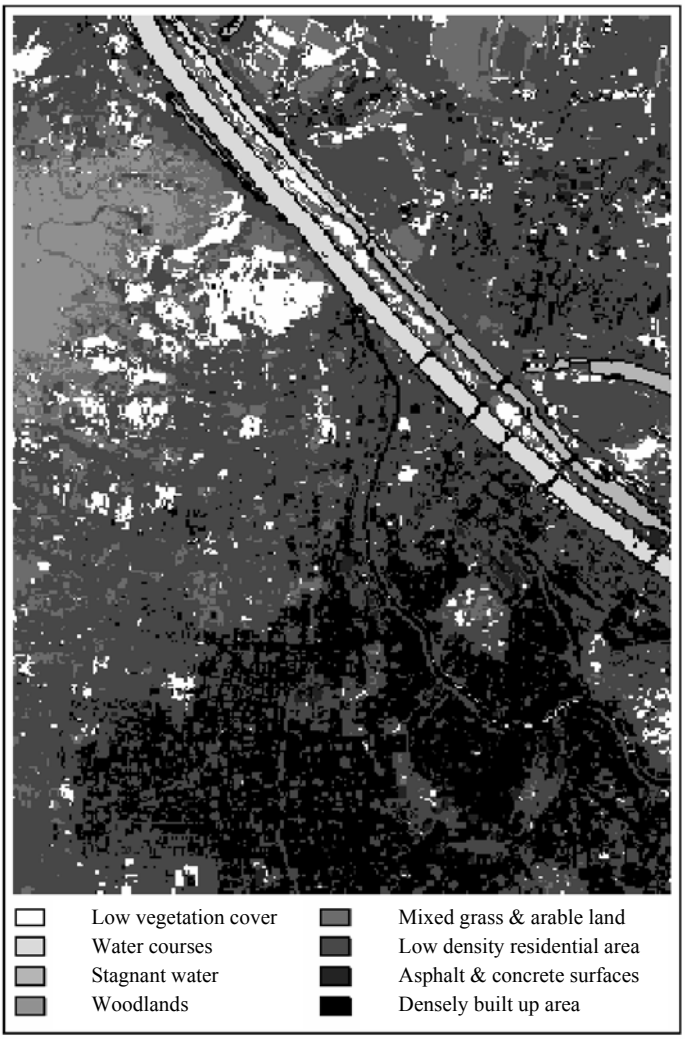


Figure 7 The fuzzy ARTMAP classified image

The primary computational difference between the fuzzy ARTMAP and the multilayer perceptron algorithms is speed. The backpropagation approach to neural network training is extremely computation-intensive, taking about one order of magnitude more time than the time for fuzzy ARTMAP, when implemented on a

serial workstation. Although this situation may be alleviated with other, more efficient training algorithms and parallel implementation, it remains one important drawback to the routine use of multi-layer perceptron classifiers. Finally, it should be mentioned that in terms of total number of pathways (i.e. the number of weight parameters) needed for the best performance, the multi-layer perceptron classifier is superior to fuzzy ARTMAP, but at the above mentioned higher computation costs and the lower classification accuracies.

7 Summary and Conclusions

Classification of terrain cover from satellite imagery represents an area of considerable current interest and research. Satellite sensors record data in a variety of spectral channels and at a variety of ground resolutions. The analysis of remotely sensed data is usually achieved by machine-oriented pattern recognition techniques of which classification based on maximum likelihood, assuming Gaussian distribution of the data, is the most widely used one. We compared fuzzy ARTMAP performance with that of an error-based learning system based upon the multi-layer perceptron and the Gaussian maximum likelihood classifier as conventional statistical benchmark on the same database. Both neural network classifiers outperform the conventional classifier in terms of map user's, map producer's and total classification accuracies. The fuzzy ARTMAP simulations did lead by far to the best out-of-sample classification accuracies, very close to maximum performance.

Evidently, the fuzzy ARTMAP classifier accommodates more easily a heterogenenous class label such as densely built-up residential areas to produce a visually and numerically correct urban land use map, even with smaller numbers of training pixels. In particular, the Gaussian maximum likelihood classifier tends to be sensitive to the purity of land use category signatures and performs poorly if they are not pure.

The study shows that the fuzzy ARTMAP classifier is a powerful tool for remotely sensed image classification. Even one epoch of fuzzy ARTMAP training yields close to maximum performance. The unique ART features such as speed and incremental learning may give the fuzzy ARTMAP multispectral image classifier the potential to become a standard tool in remote sensing especially when it comes to use data from future multichannel satellites such as the 224 channel Airborne Visible and Infrared Imaging Spectrometer (AVIRIS), and to classifiying multi-data and multi-temporal imagery or when extending the same classification to different images. To conclude the chapter, we would like to mention that the classifier leads to crisp rather than fuzzy classifications, and, thus, looses some attractivity of fuzzy pattern recognition systems. This is certainly one direction for further improving the classifier.

Acknowledgements: The authors thank Professor Karl Kraus (Department of Photogrammetric Engineering and Remote Sensing, Vienna Technical University) for his assistance in supplying the remote sensing data used in this study. This work has been funded by the Austrian Ministry for Science, Traffic and Art (funding contract no. EZ 308.937/2-W/3/95). The authors also would like to thank Petra Staufer (Wirtschaftsuniversität Wien) for her help.

References

Barnsley M. (1993): Monitoring urban areas in the EC using satellite remote sensing, *GIS Europe* 2 (8), 42-44

Benediktsson J.A., Swain P.H. and Ersoy O.K. (1990): Neural network approaches versus statistical methods in classification of multisource remote sensing data, *IEEE Transactions on Geoscience and Remote Sensing*, 28 (4), 540-552

Bezdak J.C. and Pal S.K. (eds.) (1992): *Fuzzy Models for Pattern Recognition*, IEEE Press, Piscataway [NJ]

Bischof H., Schneider W. and Pinz A.J. (1992): Multispectral classification of Landsat-images using neural networks, *IEEE Transactions on Geoscience and Remote Sensing*, 30 (3), 482-490

Carpenter G.A. (1989): Neural network models for pattern recognition and associative memory, *Neural Networks* 2 (4), 243-257

Carpenter G.A. and Grossberg S. (1995): Fuzzy ART. In: Kosko B. (ed.) *Fuzzy Engineering*, Prentice Hall, Carmel, pp. 467-497

Carpenter G.A. and Grossberg S. (eds.) (1991): *Pattern Recognition by Self-organizing Neural Networks*, MIT Press, Cambridge [MA]

Carpenter G.A. and Grossberg S. (1987a): A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics, and Image Processing* 37, 54-115.

Carpenter G.A. and Grossberg S. (1987b): ART2: Self-organization of stable category recognition codes for analog input patterns, *Applied Optics* 26 (23), 4919-4930

Carpenter G.A. and Grossberg S. (1985): Category learning and adaptive pattern recognition, a neural network model. In: *Proceedings of the Third Army Conference on Applied Mathematics and Computing*, ARO-Report 86-1, pp. 37-56

Carpenter G.A., Grossberg S. and Reynolds J.H. (1991a): ARTMAP supervised real-time learning and classification of nonstationary data by a self-organizing neural network, *Neural Networks* 4 (5), 565-588

Carpenter G.A., Grossberg S. and Rosen D.B. (1991b): Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system, *Neural Networks* 4 (6), 759-771

Carpenter G.A., Grossberg S. and Ross W.D. (1993): ART-EMAP: A neural network architecture for object recognition by evidence accumulation. In: *Proceedings of the World Congress on Neural Networks* (WCNN-93), Lawrence Erlbaum Associates, III, Hillsdak [NJ], pp. 643-656

Carpenter G.A., Grossberg S., Markuzon N., Reynolds J.H. and Rosen D. B. (1992): Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, *IEEE Transactions on Neural Networks* 3 (5), 698-713

Dawson M.S., Fung A.K. and Manry M. T. (1993): Surface parameter retrieval using fast learning neural networks, *Remote Sensing Reviews* 7 (1), 1-18

Fischer M.M., Gopal S., Staufer P. and Steinnocher K. (1994): Evaluation of neural pattern classifiers for a remote sensing application, Paper presented at the 34th European Congress of the Regional Science Association, Groningen, August 1994

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

Grossberg S. (1976a): Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors, *Biological Cybernetics* 23, 121-134

Grossberg S. (1976b): Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction and illusion, *Biological Cybernetics* 23, 187-202

Grossberg S. (1969): Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, *Journal of Mathematics and Mechanics* 19 (1), 53-91

Hara Y., Atkins R.G., Yueh S.H., Shin R.T. and Kong J. A. (1994): Application of neural networks to radar image classification, *IEEE Transactions on Geoscience and Remote Sensing* 32 (1), 100-109

Hartigan J. (1975): *Clustering Algorithms*, John Wiley, Chichester [UK], New York

Hepner G.F., Logan T., Ritter, N. and Bryant N. (1990): Artificial neural network classification using a minimal training set comparison of conventional supervised classification, *Photogrammetric Engineering and Remote Sensing* 56 (4), 469-473

Hertz J., Krogh A. and Palmer R.G. (1991): *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City [CA]

Kanellopoulos I., Wilkinson G.G. and Mégier J. (1993): Integration of neural network and statistical image classification for land cover mapping. In: *Proceedings of the International Geoscience and Remote Sensing Symposium (IGARSS'93)*, held in Tokyo, 18-21 August, Vol. 2, IEEE Press, Piscataway [NJ], pp. 511-513

Key J., Maslanik J.A. and Schweiger A.J. (1989): Classification of merged AVHRR and SMMR arctic data with neural networks, *Photogrammetric Engineering and Remote Sensing,* 55 (9), 1331-1338

McClellan G.E., DeWitt R.N., Hemmer T.H., Matheson L.N. and Moe G.O. (1989): Multispectral image-processing with a three-layer backpropagation network. In: *Proceedings of the 1989 International Joint Conference on Neural Networks*, Washington, D.C., pp. 151-153

Wilkinson G.G., Fierens F. and Kanellopoulos I. (1995): Integration of neural and statistical approaches in spatial data classification, *Geographical Systems* 2 (1), 1-20

Zadeh L. (1965): Fuzzy sets, *Information and Control* 8 (3), 338-353

Appendix A: In-Sample and Out-of-Sample Classification Error Matrices of the Classifiers

An error matrix is a square array of numbers set out in rows and columns which expresses the number of pixels assigned to a particular category relative to the actual category as verified by some reference (ground truth) data. The rows represent the reference data, the columns indicate the categorisation generated. It is important to note that differences between the map classification and reference data might be not only due to classification errors. Other possible sources of errors include errors in interpretation and delineation of the reference data, changes in land use between the data of the remotely sensed data and the data of the reference data (temporal error), variation in classification of the reference data due to inconsistencies in human interpretation etc.

Tabel A1 In-sample performance: Classification error matrices

(*a*) *Fuzzy ARTMAP*

Ground truth categories	*Classifier's categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	167	0	0	0	0	0	0	0	167
*C*2	0	285	0	0	0	0	0	0	285
*C*3	0	0	128	0	0	0	0	0	128
*C*4	0	0	0	402	0	0	0	0	402
*C*5	0	0	0	0	102	0	0	0	102
*C*6	0	0	0	0	0	293	3	0	296
*C*7	0	0	0	0	0	5	148	0	153
*C*8	0	0	0	0	0	0	0	107	107
Total	167	285	128	402	102	298	151	107	1,640

(*b*) *Multi-Layer Perceptron*

Ground truth categories	*Classifier's categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	157	10	0	0	0	0	0	0	167
*C*2	1	282	0	0	2	0	0	0	285
*C*3	0	0	128	0	0	0	0	0	128
*C*4	4	0	0	389	9	0	0	0	402
*C*5	0	0	2	2	98	0	0	0	102
*C*6	0	0	1	0	0	260	25	10	296
*C*7	0	0	0	0	0	60	93	0	153
*C*8	0	0	0	0	0	3	0	104	107
Total	162	292	131	391	109	323	118	114	1,640

(*c*) *Gaussian Maximum Likelihood*

Ground truth categories	*Classifier's categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	161	5	0	1	0	0	0	0	167
*C*2	0	284	0	0	1	0	0	0	285
*C*3	0	0	124	0	4	0	0	0	128
*C*4	0	4	0	385	13	0	0	0	402
*C*5	0	0	0	0	102	0	0	0	102
*C*6	0	0	3	0	0	214	62	17	296
*C*7	0	0	0	0	0	37	116	0	153
*C*8	0	0	0	0	0	3	0	104	107
Total	161	293	127	386	120	254	178	121	1,640

Table A2 Out-of-sample performance: Classification error matrices

(*a*) *Fuzzy ARTMAP*

Ground truth categories	*Classifier's categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	C7	*C*8	*Total*
*C*1	83	0	0	0	0	0	0	0	83
*C*2	0	142	0	0	0	0	0	0	142
*C*3	0	0	64	0	0	0	0	0	64
*C*4	0	0	0	200	0	0	0	0	200
*C*5	0	0	0	0	52	0	0	0	52
*C*6	0	0	0	0	0	146	2	0	148
*C*7	0	0	0	0	0	2	75	0	77
*C*8	0	0	0	0	0	0	0	54	54
Total	83	142	64	200	52	148	77	54	820

(*b*) *Multi-Layer Perceptron*

Ground truth categories	*Classifier's categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	C7	*C*8	*Total*
*C*1	79	4	0	0	0	0	0	0	83
*C*2	1	134	6	0	1	0	0	0	142
*C*3	0	0	64	0	0	0	0	0	64
*C*4	3	2	0	194	1	0	0	0	200
*C*5	0	3	0	0	49	0	0	0	52
*C*6	0	0	0	0	0	115	30	3	148
*C*7	0	0	0	0	0	29	48	0	77
*C*8	0	0	0	0	0	1	0	53	54
Total	83	143	70	194	51	145	78	56	820

(*c*) *Gaussian Maximum Likelihood*

Ground truth categories	*Classifier's categories*								
	*C*1	*C*2	*C*3	*C*4	*C*5	*C*6	*C*7	*C*8	*Total*
*C*1	80	3	0	0	0	0	0	0	83
*C*2	0	141	0	0	1	0	0	0	142
*C*3	0	0	62	0	1	1	0	0	64
*C*4	1	3	0	191	5	0	0	0	200
*C*5	0	5	0	0	47	0	0	0	52
*C*6	0	0	1	0	2	73	64	8	148
*C*7	0	0	0	0	0	24	53	0	77
*C*8	0	0	0	0	0	2	0	52	54
Total	81	152	63	191	56	100	117	60	820

Appendix B: *In-Sample and Out-of-Sample Map User's and Map Producer's Accuracies of the Classifiers*

Table B1 In-sample map user's and map producer's accuracies

Category	*Map user's accuracy*			*Map producer's accuracy*		
	Fuzzy ARTMAP	*Multi-layer perceptron*	*Gaussian ML*	*Fuzzy ARTMAP*	*Multi-layer perceptron*	*Gaussian ML*
*C*1	100.0	94.0	96.4	100.0	96.9	95.1
*C*2	100.0	98.9	99.6	100.0	96.9	96.9
*C*3	100.0	100.0	96.9	100.0	97.7	97.7
*C*4	100.0	96.8	95.8	100.0	99.5	97.7
*C*5	100.0	96.1	100.0	100.0	89.9	87.3
*C*6	99.0	87.8	72.3	98.3	80.5	79.8
*C*7	96.7	60.8	75.8	98.0	78.8	78.8
*C*8	100.0	97.2	97.2	100.0	91.2	85.8

Notes: Map user's accuracies for land use categories are calculated by dividing the number of correctly classified pixels in each category [i.e. the main diagonal elements of the classification error matrix] by the row totals. Map producer's accuracies for land use categories are calculated by dividing the numbers of correctly classified pixels in each category [i.e. the main diagonal elements of the classification error matrix] by the columns totals.

Table B2 Out-of-sample map user's and map producer's accuracies

Category	*Map user's accuracy*			*Map producer's accuracy*		
	Fuzzy ARTMAP	*Multi-layer perceptron*	*Gaussian ML*	*Fuzzy ARTMAP*	*Multi-layer perceptron*	*Gaussian ML*
*C*1	100.0	95.2	96.4	100.0	95.2	98.8
*C*2	100.0	94.4	99.3	100.0	93.7	92.8
*C*3	100.0	100.0	96.9	100.0	91.4	98.4
*C*4	100.0	97.0	95.5	100.0	100.0	100.0
*C*5	100.0	94.2	90.4	100.0	96.1	83.9
*C*6	98.6	77.7	49.3	98.6	79.3	73.0
*C*7	97.4	62.3	68.8	97.4	61.5	45.3
*C*8	100.0	98.1	96.3	100.0	86.9	86.7

Notes: Map user's accuracies for land use categories are calculated by dividing the number of correctly classified pixels in each category [i.e. the main diagonal elements of the classification error matrix] by the row totals. Map producer's accuracies for land use categories are calculated by dividing the numbers of correctly classified pixels in each category [i.e. the main diagonal elements of the classification error matrix] by the columns totals.

Part IV

New Frontiers in Neural Spatial Interaction Modelling

12 Neural Network Modelling of Constrained Spatial Interaction Flows

Design, Estimation and Performance Issues

with *M. Reismann and K. Hlavácková-Schindler*

In this chapter a novel modular product unit neural network architecture is presented to model singly constrained spatial interaction flows. The efficacy of the model approach is demonstrated for the origin-constrained case of spatial interaction using Austrian inter-regional telecommunication traffic data. The model requires a global search procedure for parameter estimation, such as the Alopex procedure. A benchmark comparison against the standard origin-constrained gravity model and the two-stage neural network approach, suggested by Openshaw (1998), illustrates the superiority of the proposed model in terms of the generalisation performance measured by ARV and SRMSE.

1 Introduction

The subject of spatial interaction is fundamental to economic geography and regional science. Spatial interaction models are used to facilitate the explanation and prediction of human and economic interaction over geographic space. That there have been relatively few papers in this area in recent years is merely a function of the hiatus that followed a very active period of theory development. The 1960s and 1970s saw a huge outpouring of both theoretical and empirical work. These were the heady days of Stewart and Warntz, Stouffer, Isard, Wilson and Alonso. The empiricism that emanated from their theoretical and methodological contributions filled regional science and geographical journals. The lull came not so much because interest decreased, but because very little theoretical progress has been achieved. One exception was the excitement over the work of Fotheringham on competing destinations in the early 1980s when several new models were developed and new perspectives added (Fischer and Getis 1999).

In more recent years, the major influence stems both from the emerging data-rich environment and from technological innovations. The powerful and fast computing environment now available has brought many scholars to spatial interaction theory once again, either by utilising evolutionary computation to breed novel forms of spatial interaction models (see Openshaw 1988; Turton et al. 1997) or network-based approaches to spatial interaction leading to neural spatial inter-

action models (see, for example, Gopal and Fischer 1993, Openshaw 1993, 1998, Fischer and Gopal 1994, Black 1995, Fischer et al. 1999, Bergkvist 2000, Reggiani and Tritapepe 2000, Mozolin et al. 2000). Neural spatial interaction models are termed *neural* in the sense that they have been inspired by neuroscience. But they are more closely related to conventional spatial interaction models of the gravity type than they are to neurobiological models.

In the recent past, interest has largely focused on unconstrained neural spatial interaction modelling (see, for example, Fischer et al. 1999, Fischer 2000). These models represent a rich and flexible family of spatial interaction function approximators, but they may be of little practical value if a priori information is available on accounting constraints on the predicted flows. The paper presents a novel neural network approach for the case of origin-constrained or destination-constrained spatial interaction flows. The approach is based on a modular connectionist architecture that may be viewed as a linked collection of functionally independent neural modules with identical topologies [two inputs, H hidden product units, and a single summation unit], operating under supervised learning algorithms. The prediction is achieved by combining the outcome of the individual modules using a nonlinear normalised transfer function multiplied with a bias term that implements the accounting constraint.

The efficacy of the model approach is demonstrated for the origin-constrained case by using interregional telecommunication traffic data for Austria, noisy real world data of limited record length. The Alopex procedure, a global search procedure, provides an appropriate optimisation scheme to produce least squares (LS) estimates of the model parameters. The prediction quality is measured in terms of two performance statistics, average relative variances and the standardised root mean square error. A benchmark comparison shows that the proposed model outperforms origin-constrained gravity model predictions and predictions obtained by applying the two-stage neural network approach suggested by Openshaw (1998).

The remainder of this paper is structured as follows. The next section provides some background information relevant for spatial interaction modelling, then presents the basic features of unconstrained neural spatial interaction models, and continues to discuss of how a priori information on accounting constraints can be treated from a neural network perspective. Section 3 presents the network architecture and the mathematics of the modular product unit neural network model. Moreover, it points to some crucial issues that have to be addressed when applying the model in a real world context. Section 4 is devoted to the issue of training the network model. The discussion starts by viewing the parameter estimation problem of the model as least squares learning and continues with a description of the Alopex procedure, a global search procedure, that provides an appropriate optimising scheme for LS learning. It is emphasised that the main goal of network training is to minimise the learning error while ensuring good model generalisation. The most common approach in practice is to check the network performance periodically during training to assure that further training improves generalisation as well as reduces learning error. Section 5 presents the results of a benchmark comparison of the model against the standard origin-constrained gravity model and the two-stage neural network approach that treats the prediction of flows and

the imposing of accounting constraints as two independent issues. The test bed for the evaluation uses interregional telecommunication traffic data from Austria. Section 6 summarises the results achieved, and outlines some directions for future research.

2 Background

2.1 Definitions and the Generic Interaction Model of the Gravity Type

Suppose we have a spatial system consisting of I origins and J destinations and let t_{ij} denote the volume of interaction from spatial unit (region) i to spatial unit (region) j $(i = 1, ..., I; j = 1, ..., J)$. This information may be displayed in the form of an interaction matrix of the following kind

$$\boldsymbol{T}_{I\times J} = \begin{bmatrix} t_{11} & \cdots & t_{1j} & \cdots & t_{1J} \\ \vdots & & \vdots & & \vdots \\ t_{i1} & \cdots & t_{ij} & \cdots & t_{iJ} \\ \vdots & & \vdots & & \vdots \\ t_{I1} & \cdots & t_{ij} & \cdots & t_{IJ} \end{bmatrix}. \tag{1}$$

In some cases the sets of origins and destinations are the same and, thus, $\boldsymbol{T}_{I\times J}$ is a squared matrix. The interpretation of the main diagonal of the square $\boldsymbol{T}_{I\times J}$ depends on the specific application. For instance, it might represent internal telecommunication flows within region i $(i = 1, ..., I)$. Often such values are not recorded. In other applications, for example, shopping trips from residential areas to individual shopping malls, the number of origins and destinations may differ and $\boldsymbol{T}_{I\times J}$ will not be square.

For all applications, the ith row of the matrix $\boldsymbol{T}_{I\times J}$ describes the outflows from region i to each of the J destinations, whereas inflows from each of the I origins into destination j are described by the jth column (see Equation (1)). From $\boldsymbol{T}_{I\times J}$ the volume of interaction originating from region i or terminating in region j may be calculated, that is

$$t_{i\bullet} = \sum_{j=1}^{J} t_{ij} \qquad i = 1, ..., I \tag{2}$$

and

$$t_{\bullet j} = \sum_{i=1}^{I} t_{ij} \qquad j = 1, ..., J \tag{3}$$

respectively. In turn, these marginal sums can be used to calculate the overall level of interaction that is defined as

$$t_{\bullet\bullet} = \sum_{i=1}^{I} \sum_{j=1}^{J} t_{ij}. \tag{4}$$

The Generic Interaction Model of the Gravity Type. The distribution of interactions within such a system can be described by the generic interaction model of the gravity type that asserts a multiplicative relationship between the interaction frequency and the effects of origin, destination and separation attributes, respectively. In general form it may be written as (see Wilson 1967, Alonso 1978)[1]

$$\tau_{ij} = b_{(ij)}\ r_i\ s_j\ f_{ij} \qquad i = 1, ..., I; \quad j = 1, ..., J \tag{5}$$

where τ_{ij} is the estimated flow from *i* to *j*, r_i is an origin factor characterising *i* (a measure of origin propulsiveness), s_j a destination factor characterising *j* (a measure of destination attractiveness), and f_{ij} a separation factor that measures separation from *i* to *j*. The separation factor f_{ij} is generally – but not necessarily – assumed to be a function of some univariate measure d_{ij} of separation from *i* to *j*. The exact functional form of each of these three variables is subject to varying degrees of conjecture (see Fotheringham and O'Kelly 1989). The $b_{(ij)}$ is a balancing factor with varying subscripts depending on which constraints $\{\tau_{ij}\}$ have to obey concerning $t_{i\bullet}$, $t_{\bullet j}$ or $t_{\bullet\bullet}$ (see Equations (2), (3) and (4) respectively). For example, in the origin-constrained case this conservation principle is enforced from the viewpoint of origins only, so that $b_{(i)} = t_{i\bullet} / (r_i \Sigma_{j \neq i}\ s_j\ f_{ij})$ guarantees that $t_{i\bullet} = \Sigma_j \tau_{ij}$.

Alternative forms of the general gravity model, Equation (5) can be specified by imposing different constraints on $\{\tau_{ij}\}$ (see for example, Senior 1979, Fotheringham and O'Kelly 1989, Sen and Smith 1995). In the globally constrained case the only condition specified is that the total estimated interaction $\tau_{\bullet\bullet}$ equals the total observed interaction $t_{\bullet\bullet}$. In the origin-constrained case the estimated outflows $\tau_{i\bullet}$ from each *i* have to match the observed outflows $t_{i\bullet}$ (origin constraint). In the destination-constrained case the estimated inflows $\tau_{\bullet j}$ to each region *j* must equal the observed inflows $t_{\bullet j}$ (destination constraint). Finally, in the doubly constrained case the estimated inflows $\tau_{\bullet j}$ and outflows $\tau_{i\bullet}$ have to match their observed counterparts. Note that in the unconstrained case $b_{(ij)}$ equals unity.

It is worth noting that in the origin-constrained (also called production-constrained case) the origin factor is linearly dependent on the origin-specific balancing factor $b_{(i)}$, and in the destination-constrained case (also termed attraction-constrained case) the destination factor is linearly depent on the destination-specific balancing factor $b_{(i)}$, whereas in the doubly constrained case,

[1] Alternative models based on additive adjustment formulations were introduced by Tobler (1983).

the constant of proportionality $b_{(ij)}$ depends on both origin and destination. The origin constraint and the destination constraint are isomorphic.

There are different approaches to estimating the generic spatial interaction model of Equation (5): the maximum entropy approach developed by Wilson (1967), and the loglinear approach that is a special case of Poisson regression (see for example, Aufhauser and Fischer 1985). These approaches yield identical estimates of the interaction flows in the case where the interacting units are measured on the same level of aggregation and identical sets of independent variables are used to calibrate the model.

2.2 The Classical Neural Network Approach to Spatial Interaction Modelling

The neural network approach to model spatial interactions departs from Equation (5) by viewing spatial interaction models as a particular type of input-output model. Given an input vector $\boldsymbol{x}$, the network model produces an output vector $\tilde{\boldsymbol{y}}$, say $\tilde{\boldsymbol{y}} = g(\boldsymbol{x})$. The function g is not explicitly known, but given by a finite set of samples, say $M = \{(x^u, y^u)$ with $u = 1, ..., U\}$, so that $g(x^u) = y^u$. The set M is the set of input and output vectors. The task is to find a continuous function that approximates M. In real world applications, U is generally a small number and the samples contain noise.

The Generic Neural Spatial Interaction Model. In the unconstrained case the challenge is to approximate the real-valued interaction function $g(r_i, s_j, f_{ij})$: $\Re^3 \rightarrow \Re$, where the three-dimensional euclidean real space is the input space and the one-dimensional euclidean real space the output space. In practice only bounded subsets of the spaces are considered. To approximate g, we consider the class Ω of feedforward neural network models with three input units, one hidden layer that contains H hidden units, and a single output unit. The three input units represent measures of origin propulsiveness, destination attractiveness, and spatial separation. The output unit, denoted by $\tilde{\boldsymbol{y}}$, represents the estimated flow from i to j. Formally the neural network model for the unconstrained case of spatial interaction may be written in its general form as

$$\{\Omega : \Re^3 \rightarrow \Re \,|\, \tilde{\boldsymbol{y}} = \Omega(\boldsymbol{x}, \boldsymbol{w}) = \psi\left[\gamma_0 + \sum_{h=1}^{H} \gamma_h \phi_h \left(\sum_{n=0}^{3} \beta_{hn} x_n\right)\right], \quad \boldsymbol{x} \in \Re^3; \gamma_h, \beta_{hn} \in \Re\}. \tag{6}$$

Vector $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ is the input vector augmented with a bias signal x_0 that can be thought of as being generated by a dummy unit whose output is clamped at unity. Models belonging to $\Omega(\boldsymbol{x}, \boldsymbol{w})$ may have any number of hidden units $(H = 1, 2, ...)$ with connection strengths from hidden unit h to the output unit represented by γ_h. γ_0 is a bias. The β_{hn} represent input-to-hidden connection

weights from input unit n to hidden unit h. The symbol $\boldsymbol{w} = [w_k \mid k = 1, ..., K = (5H+1)]$ is a convenient short hand notation of the $(5H+1)$-dimensional vector of all the β_{hn} and γ_h network weights and biases. ϕ_h and ψ are arbitrarily differentiable, generally nonlinear transfer functions of the hidden units and the output unit, respectively.

The Classical Unconstrained Neural Spatial Interaction Model. Hornik et al. (1989) proved that single hidden layer feedforward networks of type Equation (6) with ψ being the identity function and $\phi_h = \phi \quad (h = 1, ..., H)$ an arbitrary sigmoid transfer function can approximate any measurable function to any degree of accuracy (in appropriate metrics), given sufficiently many hidden units.[2] Thus, the neural spatial interaction model

$$\Omega_L(\boldsymbol{x}, \boldsymbol{w}) = \gamma_0 + \sum_{h=0}^{H} \gamma_h \left[1 + \exp\left(-\sum_{n=0}^{3} \beta_{hn}\, x_n \right) \right]^{-1} \tag{7}$$

represents a powerful and flexible spatial interaction function approximator. Although it has become common place to view network models such as those of Equations (6) or (7) as types of black boxes, this leads to inappropriate applications that may fail not because such network models do not work well but because the issues are not well understood. Failures in applications can often be attributed to inadequate learning (training), inadequate numbers of hidden units, or the presence of a stochastic rather than a deterministic relation between input and target.

Least Squares Learning. If we view Equation (7) as generating a family of approximators (for example, as $\boldsymbol{w}$ ranges over $\boldsymbol{W}$) to some specific empirical spatial interaction phenomenon relating inputs $\boldsymbol{x}$ to some response $\boldsymbol{y}$, then we need a way to pick the best approximation from this family. This is the function of learning in the context of neural network modelling. The goodness of the approximation can be evaluated using an error (penalty) function that measures how well the model output $\tilde{\boldsymbol{y}}$ matches the target output $\boldsymbol{y}$ corresponding to given input $\boldsymbol{x}$. The penalty should be zero when target and model output match, and positive otherwise. A leading case is the least squares error function. With this error function, learning must arrive at $\boldsymbol{w}^*$ that solves

$$\min_{\boldsymbol{w} \in \boldsymbol{W}} (E\{\tfrac{1}{2}[\boldsymbol{y} - \Omega_L(\boldsymbol{x}, \boldsymbol{w})]^2\}) = E\{\tfrac{1}{2}[\boldsymbol{y} - E(\boldsymbol{y} \mid \boldsymbol{x})^2]\} + E\{\tfrac{1}{2}[E(\boldsymbol{y} \mid \boldsymbol{x}) - \Omega_L(\boldsymbol{x}, \boldsymbol{w})]^2\}$$

where E denotes the expectation operator. Finding $\boldsymbol{w}^*$ is precisely the problem of determining the parameters of an optimal least squares approximator to $E(\boldsymbol{y} \mid \boldsymbol{x})$,

[2] Sigmoid transfer functions are somewhat better behaved than many other functions with respect to the smoothness of the error surface. They are well behaved outside of their local region in that they saturate and are constant at zero or unity outside the training region. Sigmoidal units are roughly linear for small weights (net input near zero) and are increasingly nonlinear in their response as they approach their points of maximum curvature on either side of the midpoint.

the conditional expectation of $\boldsymbol{y}$ given $\boldsymbol{x}$. The expectation defining the optimand is unknown, so that this problem has to be solved statistically.

Backpropagation of Gradient Descent Errors. Backpropagation of gradient descent errors is such a method that allows parameters to be learned from experience in a process that resembles trial and error (see Rumelhart et al. 1986). Experience is based on empirical observations on the phenomenon of spatial interaction of interest. Thus, we assume a training set consisting of observations x^u $(u = 1, ..., U)$ on the input variables together with observations y^u $(u = 1, ..., U)$ on corresponding target variables, the network model is to learn to associate with x^u. According to the backpropagation of the gradient descent errors procedure one starts with a set of random weights, say $\boldsymbol{w}_0$, and then updates them by the following formula for the *u*th step

$$\boldsymbol{w}(u) = \boldsymbol{w}(u-1) + \eta \nabla \Omega_L[x^u, \boldsymbol{w}(u-1)]\{y^u - \Omega_L[x^u, \boldsymbol{w}(u-1)]\}$$

where η is a learning rate and $\nabla \Omega_L$ denotes the gradient of Ω_L with respect to $\boldsymbol{w}$. The weights are adjusted in response to errors in hitting the target where errors are measured in terms of the least squares error function. This error is propagated back. Although many modifications of and alternatives to this parameter estimation approach have been suggested in the neural network literature over the past years, experience shows that surprisingly good network model performance can often be achieved with the epoch-based stochastic version of this learning approach (see Fischer and Gopal 1994).

2.3 Departure from the Classical Neural Network Approach

Although classical neural spatial interaction models of type (6) represent a rich and flexible family of spatial interaction function approximators for real world applications, they may be of little practical value in situations where the set of row totals or the set of column totals of the spatial interaction matrix $\boldsymbol{T}_{I \times J}$ is known a priori. For such situations, the families of production-constrained and attraction-constrained models of the gravity type had been developed.

The question arises how to build neural network models for the case of singly constrained spatial interaction flows. Following Openshaw (1998) constrained spatial interaction modelling may be viewed as consisting of two parts: (i) the prediction of flows, and (ii) the imposition of accounting constraints.

These two parts can be treated separately or simultaneously. The question that follows is whether to embed the constraint-handling mechanism within the neural network approach (a one-stage modelling approach) or whether to estimate the unconstrained neural spatial interaction model first and then to apply the accounting constraints subsequently (a two-stage modelling approach). The one-stage modelling approach is more difficult because it requires major changes to the model

structure, whereas the two-stage approach is much simpler (for an application see Mozolin et al. 2000).

3 The One-Stage Modelling Approach: The Modular Product Unit Network Model

3.1 Why Product Rather than Summation Unit Networks?

Conventional neural spatial interaction models, such as $\Omega_L(\boldsymbol{x},\boldsymbol{w})$, are constructed using a single hidden layer of summation units. In these networks each input to the hidden node is multiplied by a weight and then summed. A nonlinear transfer function, such as the logistic function, is used to squash the sum. Neural network approximation theory has shown the attractivity of such summation networks for unconstrained spatial interaction contexts.

In the neural network community it is well known that supplementing the inputs to a neural network model with higher-order combinations of the inputs increases the capacity of the network in an information capacity sense (see Cover 1965) and its ability to learn (see Giles and Maxwell 1987). Although the error surface of product unit networks contains more local minima than when using summation unit networks, the surface is locally smooth. However, the price to be paid is a combinatorial explosion of higher-order terms as the number of inputs to the network increases.

The product units introduced by Durbin and Rumelhart (1989) attempt to make use of the above fact. Product unit networks have the advantage that – given an appropriate training algorithm – the units can learn the higher-order terms that are required to approximate a specific constrained spatial interaction function. This property motivates the use of the product unit rather than the standard summation unit neural framework for modelling singly constrained interaction flows over space.

3.2 The Network Architecture

Product units compute the product of inputs, each raised to a variable power. They can be used in a network in many ways, but the overhead required to raise an arbitrary base to an arbitrary power makes it more likely that they will supplement rather than replace summation units (Durbin and Rumelhart 1989).[3] Thus, we use the term *product unit networks* (or product networks) to refer to networks containing both product and summation units.

---

[3] Note that product units are vulnerable to translation and rotation of the input space in the sense that a learnable problem may no longer be learnable after translation. Using a number of product units in parallel can help compensate for rotational and translational vulnerability of single product units.

Figure 1 illustrates the modular network architecture of the product unit neural network that we propose to model the origin-constrained case of spatial interactions. Here, modularity is seen as decomposition on the computational level. The network is composed of two processing layers and two layers of network parameters. The first processing layer is involved with the extraction of features from the input data. This layer is implemented as a layer of J functionally independent modules with identical topologies. Each module is a feedforward network with two inputs x_{2j-1} and x_{2j} representing measures of destination attractiveness and separation between origin and destination, respectively; H hidden product units $[(j-1)H+1, ..., (j-1)H+h, ..., jH]$, denoted by the symbol Π; and terminates with a single summation unit, denoted by the symbol Σ. The collective outputs of these modules constitute the input to the second processing layer consisting of J output units that perform the flow prediction and enforce satisfactorily the conservation rule of interaction from the viewpoint of origins.[4]

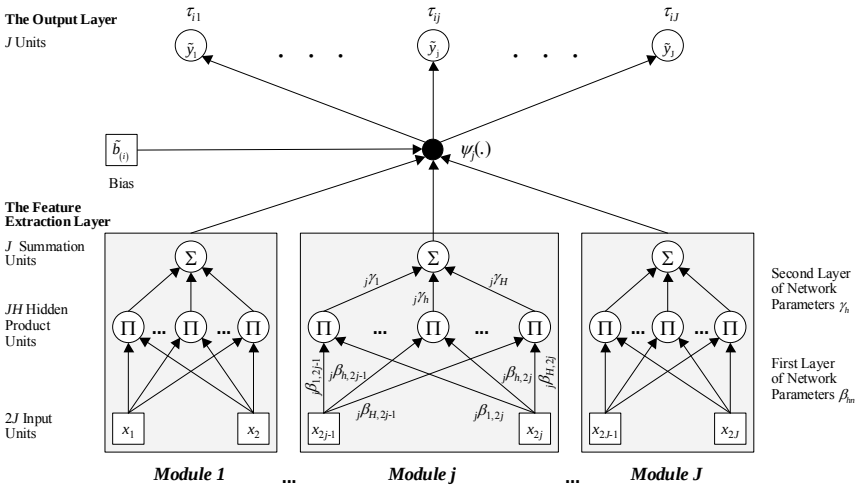


Figure 1 Architecture of the product unit neural spatial interaction model: The origin-constrained case

The first layer of network parameters includes $2JH$ connection weights, so that

$$ {}_1\boldsymbol{w} = ({}_j\beta_{(j-1)H+1,2j-1}, \cdots, {}_j\beta_{(j-1)H+h,2j-1}, \cdots, {}_j\beta_{jH,2j-1}, {}_j\beta_{(j-1)H+1,2j}, \cdots, {}_j\beta_{(j-1)H+h,2j}, \cdots, {}_j\beta_{jH,2j}) $$

and the second layer contains JH weights

[4] In the attraction-constrained case the conservation principle is enforced from the viewpoint of destinations only.

$${}_2\boldsymbol{w} = ({}_j\gamma_{(j-1)H+1}, \ldots, {}_j\gamma_{(j-1)H+h}, \ldots, {}_j\gamma_{jH}) .$$

We have incorporated the basic trick of weight sharing into our network design to reduce model complexity. Weight sharing involves forcing the set of connection weights to be identical across the J modules. Thus, $\boldsymbol{w} = ({}_1\boldsymbol{w}, {}_2\boldsymbol{w})$ is a $(3H)$-dimensional rather than a $(3JH)$-dimensional vector. Consequently, notation may be simplified as follows for all $j = 1, ..., J$: $(j-1)H + h$ by definition is h, jH by definition is H, ${}_j\beta_{(j-1)H+h,2j-1} =: \beta_{h,2j-1}$, ${}_j\beta_{(j-1)H+h,2j}$ by definition is $\beta_{h,2j}$, ${}_j\gamma_{(j-1)H+h} =: \gamma_h$.

3.3 Mathematical Description

The network architecture described above implements the general class of neural models of origin-constrained spatial interaction

$${}^{\pi}\Omega_{orig}(\boldsymbol{x}, \boldsymbol{w})_j = \psi_j \left[\sum_{h=1}^{H} \gamma_h \; \phi_h \left(\prod_{n=2j-1}^{2j} x_n^{\beta_{hn}} \right) \right] \qquad j = 1, ..., J$$

with $\phi_h : \Re \to \Re$, $\psi_j : \Re \to \Re$ and a $(2J)$–dimensional vector $\boldsymbol{x} = (x_1, x_2, ..., x_{2j-1}, x_{2j}, ..., x_{2J-1}, x_{2J})$ where x_{2j-1} represents a variable s_j pertaining to destination j $(j = 1, ..., J)$ and x_{2j} represensts a variable f_{ij} pertaining to the separation from region i to region j $(i = 1, ..., I;\ j = 1, ..., J)$ of the spatial interaction system under scrutiny. β_{hn} $(h = 1, ..., H;\ n = 2j-1, 2j)$ are the input-to-hidden connection weights and γ_h $(h = 1, ..., H)$ are the hidden-to-output weights in the jth module of the network model. The symbol $\boldsymbol{w}$ is a convenient shorthand notation of the $(3H)$-dimensional vector of all the model parameters. ψ_j $(j = 1, ..., J)$ represents a nonlinear summation unit and ϕ_h $(h = 1, ..., H)$ a linear hidden product unit transfer function.

We restrict our attention to some specific members of the above class. First, we assume the hidden transfer function $\phi_h(\cdot)$ to be the identity function. Thus, the output of the hth hidden product unit of the jth network module, denoted by ${}_jz_h$, reads as

$${}_jz_h = \phi_h({}_jnet_h) = \prod_{n=2j-1}^{2j} x_n^{\beta_{hn}} \qquad j = 1, ..., J;\ h = 1, ..., H.$$

Second, we assume that each output unit, ${}^{\pi}\Omega_{orig}(\boldsymbol{x}, \boldsymbol{w})_j$ by definition is $\tilde{y}_j\,(j = 1, ..., J)$, uses a nonlinear normalised transfer function $\psi_j(\cdot)$ so that

$$ {}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})_j = \psi_j(\text{net } j) = \tilde{b}_{(i)} \frac{\text{net } j}{\sum_{j'=1}^{J} \text{net } j'} \qquad j = 1, ..., J \tag{8} $$

augmented by a bias unit $\tilde{b}_{(i)}$, where net j is the value of the weighted sum for the jth output node given by

$$ \text{net } j = \sum_{h=1}^{H} \gamma_h \prod_{n=2j-1}^{2j} x_n^{\beta_{hn}} \qquad j = 1, ..., J. $$

The choice of the output transfer function of Equation (8) is motivated by the goal to ensure that the network outputs satisfy the conservation principle (Ledent 1985) that is enforced from the viewpoint of origins: $\tilde{b}_{(ij)} = \tilde{b}_{(i)}$. The ${}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})_j$ may be interpreted as probability of spatial interactions, conditioned on the output $\Sigma_{h=1\ j}^{H} z_h$ of the jth network module.

With the two above specifications, the modular product unit neural network to model origin-constrained spatial interactions reads in a compact way as

$$ {}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})_j = \tilde{b}_{(i)} \frac{\sum_{h=1}^{H} \gamma_h \prod_{n=2j-1}^{2j} x_n^{\beta_{hn}}}{\sum_{j'=1}^{J} \sum_{h'=1}^{H} \gamma_{h'} \prod_{n=2j'-1}^{2j'} x_n^{\beta_{h'n}}} \qquad j = 1, ..., J \tag{9} $$

where $\tilde{b}_{(i)}$ is the bias signal that can be thought as being generated by a 'dummy unit' whose output is clamped at the scalar $t_{i\bullet}$. The relation of Equation (9) to the generic spatial interaction model (5) becomes evident when setting $x_{2j-1} = s_j$ and $x_{2j} = f_{ij}$

$$ \tau_{ij} := {}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})_j = \tilde{b}_{(i)} \frac{\sum_{h=1}^{H} \gamma_h\ s_j^{\beta_{h1}}\ f_{ij}^{\beta_{h2}}}{\sum_{j'=1}^{J} \sum_{h'=1}^{H} \gamma_{h'}\ s_{j'}^{\beta_{h'1}}\ f_{ij'}^{\beta_{h'2}}} \qquad j = 1, ..., J. $$

Analogously one arrives at the modular product unit neural network for the destination-constrained case, that is

$$ {}^{\pi}\Omega_{dest}(\boldsymbol{x},\boldsymbol{w})_i = \tilde{b}_{(j)} \frac{\sum_{h=1}^{H} \gamma_h \prod_{n=2i-1}^{2i} x_n^{\beta_{hn}}}{\sum_{i'=1}^{I} \sum_{h'=1}^{H} \gamma_{h'} \prod_{n=2i'-1}^{2i'} x_n^{\beta_{h'n}}} \qquad i = 1, ..., I \tag{10} $$

where $\tilde{b}_{(j)}$ is the bias signal that can be thought as being generated by a 'dummy unit' whose output is clamped at the scalar $t_{\bullet j}$. Setting $x_{2j-1} = r_i$ and $x_{2j} = f_{ij}$ then Equation (10) becomes[5]

$$\tau_{ij} := {}^{\pi}\Omega_{dest}(\boldsymbol{x}, \boldsymbol{w})_i = \tilde{b}_{(j)} \frac{\sum_{h=1}^{H} \gamma_h \, r_i^{\beta_{h1}} \, f_{ij}^{\beta_{h2}}}{\sum_{i'=1}^{I} \sum_{h'=1}^{H} \gamma_{h'} \, r_{i'}^{\beta_{h'1}} \, f_{i'j}^{\beta_{h'2}}} \qquad i = 1, ..., I.$$

3.4 Two Issues of Crucial Importance for Real World Applications

Two major issues have to be addressed when applying spatial interaction models ${}^{\pi}\Omega$ in a real world context: first, the issue of finding a suitable number H of hidden product units (the so-called representation problem); and second, the issue of network training or learning (the so-called learning problem). The first issue is a challenging task because the number of hidden product units affects the generalisation performance of the model. Small networks learn slowly or may not learn at all to an acceptable error level. Larger networks usually learn better, but such sizes may lead to generalisation degradation known as overtraining or overfitting. The correct number of hidden units is problem dependent and a function of the complexity of the input-output mapping to be realised by the model and the required accuracy. The common goal is to simplify the model in terms of H without sacrificing its generalisation performance.

Various techniques have been developed in the neural network literature to control the effective complexity of neural network models, in most cases as part of the network training process itself. The maximum complexity of our models can be controlled by limiting the number of hidden units, so one obvious approach to the bias-variance trade-off is to train several model candidates with different numbers of hidden product units, and to select that model that gives the best generalisation performance. An obvious drawback of such an approach is its trial and error nature.

A more principled approach to the problem that has been used by Fischer and Gopal (1994) is *stopped* or *cross-validation training*. With this approach, an overparametrised model (larger H) is trained until the error on further independent data (the validation set) deteriorates, then training is stopped. This contrasts to the previous approach because the choice of H does not require convergence of the training process. The training process is used to perform a directed search in the parameter space for a model that does not overfit the data and, thus, demonstrates generalisation performance. However, this approach has shortcomings also. First, in practice it may be difficult to identify when to stop training. Second, the results may depend on the specific training set-validation set pair chosen. Third, the mo-

[5] Note that this holds only if $I = J$.

del that has the best performance on the validation set is not necessarily the one with the best performance on the test set. The second issue involves network training or learning (i.e., parameter estimation). This issue will be addressed in more detail in the following section.

4 Training the Modular Product Unit Network Model

4.1 The Optimisation Problem

Having identified the model structure for singly constrained spatial interaction prediction in the previous section, we can now follow Section 2 to view network training in an optimisation context and proceed by considering the parameter estimation problem as least squares learning. The goal of least squares learning is to find $\boldsymbol{w}^*$ so that the least squares error function, say Q, is minimised:[6]

$$\min_{\boldsymbol{w}} Q(x^{u1}, y^{u1}, \boldsymbol{w}) = \min_{\boldsymbol{w}} \sum_{(x^{u1}, y^{u1}) \in M_1} [y^{u1} - {}^{\pi}\Omega(x^{u1}, \boldsymbol{w})]^2$$

where M_1 denotes the training set consisting of observations, say $x^{u1} (u1 = 1, ..., U_1)$, on the input variables together with observations on corresponding target variables, the network model is to learn to associate with x^{u1}.

$Q(x^{u1}, y^{u1}, \boldsymbol{w})$ is nonnegative, continuously differentiable on the $(3H)$-dimensional parameter space, which is a finite dimensional closed bounded domain and, thus, compact. The compactness of the parameter space is of great theoretical convenience. It can be shown that $Q(x^{u1}, y^{u1}, \boldsymbol{w})$ assumes its value $\boldsymbol{w}^*$ as the weight minimum under certain conditions, but characteristically there exist many minima in real world applications all of which satisfy

$$\nabla Q(x^{u1}, y^{u1}, \boldsymbol{w}^*) = 0$$

where ∇Q denotes the gradient of Q. The minimum for which the value of Q is smallest is termed the global minimum and other minima are called local minima. Unfortunately there is no guarantee about the type of minimum is encountered.

The fraction of the parameter space that contains a solution depends on the capacity of the network model (in an information theoretic sense) and the complexity of the problem at hand. Given the mountainous error surface that is characteristic of product unit networks, a local search algorithm such as backpropagation of gradient descent errors is ineffective and usually converges to local

---

[6] Bishop (1995) has shown that the least squares error function can be derived from the principle of maximum likelihood on the assumption of Gaussian distributed target data. Of course, the use of the error function does not require the target data to have a Gaussian distribution.

minima. In contrast, global search algorithms such as the Alopex procedure have heuristic strategies to help escape from local minima.

The success of global search procedures in finding a global minimum of a given function such as $Q(x^{u1}, y^{u1}, \boldsymbol{w})$ over $\boldsymbol{w} \in \boldsymbol{W}$ hinges on the balance between an exploration process, a guidance process, and a convergence-inducing process (Hassoun 1995). The exploration process gives the search a mechanism for sampling a sufficiently diverse set of parameters $\boldsymbol{w}$ in $\boldsymbol{W}$. The Alopex procedure performs an exploration process that is stochastic in nature.[7] The guidance process is an implicit process that evaluates the relative quality of search points (i.e., two consecutive search points) and uses correlation guidance to move towards regions of higher-quality solutions in the parameter space. Finally, the convergence-inducing process ensures the convergence of the search to find a fixed solution $\boldsymbol{w}^*$. The convergence-inducing process is realised effectively by a parameter T, called temperature in analogy to the simulated annealing procedure, that is gradually decreased over time. The dynamic interaction among these three processes is responsible for giving the Alopex search process its global optimising character.

4.2 The Alopex Procedure

Consider training data set $M_1 = \{(x^{u1}, y^{u1})$ with $u1 = 1, ..., U_1\}$, We assume that the data were generated by some true underlying function $g(\boldsymbol{x})$. Our objective is to learn the parameter $\boldsymbol{w} = (w_k \mid k = 1, ..., 3H)$ of the approximating function ${}^{\pi}\Omega(\boldsymbol{x}, \boldsymbol{w})$ whose form is dependent upon the choice of H.

Alopex is a correlation-based method for solving the parameter estimation problem (see Bia 2000; Unnikrishnan and Venugopal 1994; Harth and Pandya 1988). The error function Q is minimised by means of weight changes that are calculated for the nth step ($n > 2$) of the iteration process in batch mode as follows:[8]

$$w_k(n) = w_k(n-1) + \delta \, \mathrm{sgn}[\varepsilon - p_k(n)]$$

where δ is the step size that has to be chosen a priori and ε is uniformly distributed random value with $\varepsilon \in [0,1]$. The probability of change of the parameter is calculated as

$$p_k(n) = \left[1 + \exp\left\{ -\frac{C_k(s)}{T(s)} \right\} \right]^{-1}$$

with $C_k(n)$ given by the correlation

[7] Alopex is an acronym for **al**gorithm f**o**r **p**attern **ex**traction.

[8] For the first two iterations, the weights are chosen randomly.

$$C_k(n) = [w_k(n-1) - w_k(n-2)]\{Q[\boldsymbol{x}, \boldsymbol{y}, w_k(n-1)] - Q[\boldsymbol{x}, \boldsymbol{y}, w_k(n-2)]\}$$
$$= [\Delta w_k(n)]\{\Delta Q[\boldsymbol{x}, \boldsymbol{y}, w_k(n)]\}.$$

The weight will be incremented in a given fixed magnitude δ when Δw_k is greater than zero and the opposite when Δw_k is less than zero. The sign of C_k indicates whether Q varies in the same way as w_k. If $C_k > 0$, both Q and w_k will be raised or lowered. If $C_k < 0$, one will be lowered and the other raised.

If T is too small, the algorithm gets trapped into local minima of Q. Thus, the value of T for each iteration $T(n)$ is chosen using the following heuristic annealing schedule

$$T(n) = \begin{cases} \frac{\delta}{3HN} \sum_k \sum_{n'=n-N}^{n-1} |C_k(n')| & \text{if } n \text{ is a multiple of } N \\ T(n-1) & \text{otherwise} \end{cases} \tag{11}$$

where $3H$ denotes the number of weights. The annealing schedule controls the randomness of the algorithm. When T is small, the probability of changing the parameters is around zero if C_k is negative and around unity if C_k is positive. If T is large, then $p_k \cong 0.5$. This means that there is the same probability to increment or decrement the weights and that the direction of the steps is now random. In other words, high values of T imply a random walk, whereas low values cause a better correlation guidance (see Bia 2000). The effectiveness of Alopex in locating global minima and its speed of convergence critically depend on the balance of the size of the feedback term $\Delta w_k \, \Delta Q$ and the temperature T. If T is very large compared to $\Delta w_k \, \Delta Q$ the process does not converge. If T is too small, a premature convergence to a local minimum might occur.

Initial Values. The algorithm has three parameters: the initial temperature T, the number of iterations N over which the correlations are averaged for annealing, and the step size δ. The temperature T and the N-iterations cycles seem to be of secondary importance for the final performance of the algorithm. The initial temperature T may be set to a large value of about 1,000. This allows the algorithm to get an estimate of the average correlation in the first N iterations and reset it to an appropriate value according to Equation (11). N may be chosen between 10 and 100. In contrast to T and N, δ is a critical parameter that has to be selected with care. There is no way to a priori identify δ in the case of multimodal error surfaces.

The Termination Criterion. An important issue associated with network training is the termination criterion. The main goal of training is to minimise the learning error while ensuring good model generalisation. It has been observed that forceful training may not produce network models with adequate generalisation ability, although the learning error achieved is small. The most common remedy

for this problem is to monitor model performance during training to assure that further training improves generalisation as well as reduces learning error. For this purpose an additional set of validation data $M_2 = \{(x^{u2}, y^{u2})$ with $u2 = 1,...,U_2\}$ independent from the training data is used.

In a typical training phase, it is normal for the validation error to decrease. However, this trend may not be permanent. At some point the validation error usually reverses or its improvement is extremely slow. Then the training process should be stopped. In our implementation of the Alopex procedure network training is stopped when $\kappa = 40,000$ consecutive iterations are unsuccessful. κ has been chosen so large at the expense of the greater training time, to ensure more reliable estimates. Of course, setting the number of unsuccessful iterations to 40,000 (or more) does not exclude the possibility that there would be successful steps ahead if training continued. At some stage a training algorithm may recover from some local attractor and accomplish further error minimisation, but we require that it should occur within a certain number of iterations. Obviously, when training is stopped, the final set of network weights does not correspond to the best result found. Thus, it is necessary to store the parameter values in a separate array every time a successful training step is made. At the end of the training process the best set of parameter values is then recalled.

5 Benchmark Comparisons

The attraction of the approach suggested to model the case of singly constrained spatial interaction depends not only on the awareness of what it can offer, but also on empirical illustrations of what can be gained in comparison to alternative model approaches. The standard origin-constrained gravity model and the two-stage neural network approach, suggested by Openshaw (1998) and implemented by Mozolin et al. (2000), are used as benchmark models. All three models are estimated by means of the Alopex procedure to eliminate the effect of different estimation procedures on the result. In order to do justice to each model, the δ parameter is systematically sought for each model.

5.1 Performance Measures

The ultimate goal of any function approximator is its usefulness to generate accurate out-of-sample prediction. One way to assess directly the generalisation ability is to measure how well the approximator predicts the flows for new input data that were not used to fit the model. For this purpose some performance measure is required. One needs to be very careful when selecting a measure to compare different models. A comparison may become meaningless if the performance measure has been used to estimate the parameters in the case of one model, but not in the case of the others. There is some literature that ignores this. In this study model

performance is measured on the testing (prediction, out-of-sample) data set, say $M_3 = \{(x^{u3}, y^{u3})$ with $u3 = 1,..,U_3\}$, by means of two least squares related overall performance measures. The first performance measure are the average relative variances, $\mathrm{ARV}(M_3)$, a normalised mean squared error metric that is used widely in the neural network community

$$\mathrm{ARV}(M_3) = \frac{\sum_{(x^{u3}, y^{u3}) \in M_3} [y^{u3} - {}^{\pi}\Omega(x^{u3}, \boldsymbol{w})]^2}{\sum_{(x^{u3}, y^{u3}) \in M_3} (y^{u3} - \overline{y}^{u3})^2}$$

$$= \frac{1}{\sigma^2 U_3} \sum_{(x^{u3}, y^{u3}) \in M_3} [y^{u3} - {}^{\pi}\Omega(x^{u3}, \boldsymbol{w})]^2$$

where (x^{u3}, y^{u3}) denotes the $u3$th pattern of the testing set M_3, $\overline{y}^{u3}$ the average over the U_3 desired values. The averaging, that is the division by U_3 (the number of patterns in M_3), makes ARV independent of the size of M_3. The division by the estimated σ^2 of the data removes the dependence on the dynamic range of the data. This implies that if the estimated mean of the observed data would be taken as predictor, ARV would equal unity (Weigend et al. 1991). The statistic has a lower limit of zero indicating perfectly accurate predictions and an upper limit that in practice is unity.[9]

The second performance measure is the standardised root mean square error (SRMSE) that is used widely by spatial analysts (see Fotheringham and Knudsen 1987)

$$\mathrm{SRMSE}(M_3) = \frac{1}{\overline{y}^{u3}} \left[\sum_{(x^{u3}, y^{u3}) \in M_3} [y^{u3} - {}^{\pi}\Omega(x^{u3}, \boldsymbol{w})]^2 / U_3 \right]^{\frac{1}{2}}.$$

This statistic – closely related to the ARV-measure – has a lower limit of zero indicating perfectly accurate predictions and an upper limit that is variable and depends on the distribution of the y^{u3}.[10]

5.2 The Data

The test bed for the benchmark comparisons uses interregional telecommunication traffic data for Austria. From three Austrian data sources – a (32, 32)-interregional telecommunication flow matrix (t_{ij}), a (32, 32)-distance matrix (d_{ij}), and gross regional products for the 32 telecommunication regions – a set of 992 3-tupel

[9] ARV-values greater than unity occur if the average error is greater than the mean.

[10] SRMSE-values greater than unity occur if the average error is greater than the mean.

(s_j, d_{ij}, t_{ij}) with $i, j = 1,...,32$ $(i \neq j)$ is constructed. The first two components represent the input variables x_{2j-1} and x_{2j} of the *j*th module of the network model ${}^{\pi}\Omega_{orig}(\boldsymbol{x}, \boldsymbol{w})$, and the last component the target output. The bias term $\tilde{b}_{(i)}$ is clamped to the scalar $t_{i\bullet}$. s_j represents the potential draw of telecommunication in *j* and is measured in terms of the gross regional product, d_{ij} in terms of distances from *i* to *j*, while t_{ij} and $t_{i\bullet}$ represent telecommunication traffic flows. The input data are preprocessed to data scaled into [0.1, 0.9].[11]

The telecommunication traffic flow data stem from network measurements of carried traffic in Austria in 1991, in terms of erlang, an internationally widely used measure of telecommunication contact intensity, which is defined as the number of phone calls (including facsimile transfers) multiplied by the average length of the call (transfer) divided by the duration of measurement (for more details, see Fischer and Gopal 1994). [12] The data refer to the telecommunication traffic between the 32 telecommunication districts representing the second level of the hierarchical structure of the Austrian telecommunication network (see Figure 2). Intraregional traffic (i.e. *i* = *j*) is not considered due to measuremet problems.

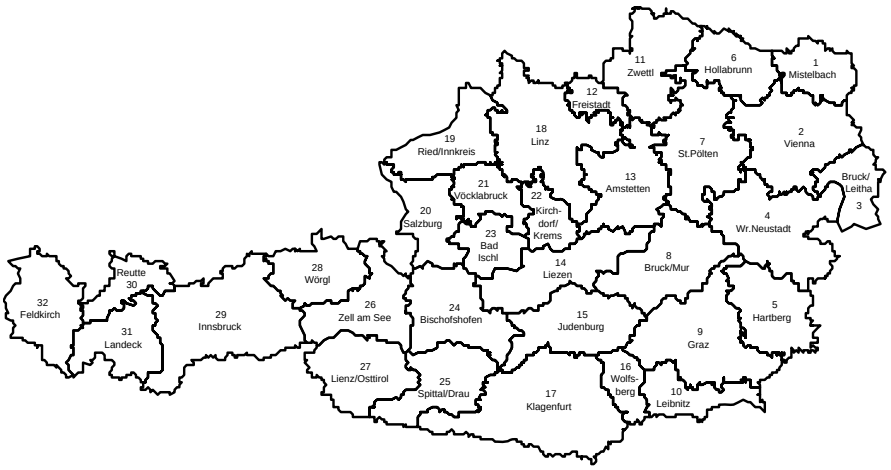


Figure 2 The regional system for modelling interregional telecommunication traffic in Austria

One of the simplest methods for estimating the prediction error is data splitting. This method simulates model validation with new data by partitioning the total data set of 992 samples into three subsets: (i) the training (in-sample) set $M_1 = \{(x^{u1}, y^{u1})$ with $u1 = 1,...,U_1 = 496$ patterns$\}$; (ii) the internal validation set $M_2 = \{(x^{u2}, y^{u2}$ with $u2 = 1,...,U_2 = 248$ patterns$\}$; and (iii) the testing (pre-

[11] Except for the standard origin-constrained model.

[12] Note that flows are generally measured as discrete counts, but in this study in terms of erlang, a metric variable.

diction, out-of-sample) set $M_3 = \{(x^{u3}, y^{u3} \text{ with } u3 = 1,...,U_3 = 248 \text{ patterns}\}$.[13] M_1 is used only for parameter estimation, whereas M_2 is used for validation. The generalisation performance of the model is assessed on the testing set M_3.

Table 1 Descriptive statistics: The training, validation and testing sets

Variables	*Mean*	*Standard deviation*	*Minimum*	*Maximum*
Whole set M				
s_j	26,364,563	50,350,660	2,310,400	285,193,984
d_{ij}	229.4	124.6	30.0	630.0
t_{ij}	8.6	22.6	0.0	257.9
$t_{i\bullet}$	266.0	350.1	41.9	1830.1
Training set M_1				
s_j	26,142,923	49,711,907	2,310,400	285,193,984
d_{ij}	234.1	129.6	35.0	630.0
t_{ij}	9.6	26.2	0.0	257.9
$t_{i\bullet}$	297.0	429.1	41.9	1830.1
Validation set M_2				
s_j	26,517,946	50,891,071	2,310,400	285,193,984
d_{ij}	219.3	121.4	30.0	590.0
t_{ij}	7.1	16.6	0.0	166.8
$t_{i\bullet}$	220.9	221.4	45.6	759.8
Testing set M_3				
s_j	26,654,459	51,069,577	2,310,400	285,193,984
d_{ij}	230.3	116.7	37.0	627.0
t_{ij}	8.0	19.7	0.0	195.2
$t_{i\bullet}$	249.0	262.5	55.3	895.7

Note: M consists of 992 patterns, M_1 of 496 patterns, M_2 of 248 patterns, and M_3 of 248 patterns.

Though the simplicity of this method is appealing, an obvious concern is the necessary reduction in the amount of training data. In deciding how to partition the data, a compromise has to be made between creating a test set large enough to fully test the fitted model while still retaining a sufficient amount of training and internal validation data. If the test set is too small then the variance of the prediction error estimate will be high due to the small sample size. Even though random splits are commonly used and appear to work reasonably well in the case of unconstrained spatial interaction, a more systematic splitting method has to be used

[13] This static approach for evaluating the performance of a neural model has been used for many years in the connectionist community in general and in neural spatial interaction modelling in particular (see, for example, Fischer and Gopal 1994). Recent experience has found this approach to be rather sensitive to the specific splitting of the data. Fischer and Reismann (2002) suggest an approach that overcomes this issue of fixed data splitting by combining the purity of splitting the data with the power of bootstrapping, (see also Fischer and Reismann 2002a).

in the case of constrained spatial interaction to guarantee that outflows $\{t_{i1}, ..., t_{iJ}\}$ from a region *i* are not split.[14]

Some descriptive statistics characterising M_1, M_2 and M_3 are summarised in Table 1. As can be seen from this table there are no large differences between the training, validation, and test sets. Nevertheless, there are substantial differences that will present some challenge to the estimation procedure used, especially in t_{ij} and $t_{i\bullet}$.

5.3 Model Estimation and the Overfitting Problem

Deciding on an appropriate number *H* of product units and on the value for the Alopex parameter δ (the step size) is somewhat discretionary, involving the familiar trade-off between speed and accuracy. The approach adopted for this evaluation is stopped (cross-validation) training. The Alopex-parameters *T* and *N* are set to 1,000 and 10, respectively.

It is worth emphasising that the training process is sensitive to its starting point. Despite recent progress in finding the most appropriate parameter initialisation that would help Alopex to find near optimal solutions, the most widely adopted approach still uses random weight initialisation in order to reduce fluctuation in evaluation. Each experiment to determine *H* and δ is repeated 60 times, the model being initialised with a different set of random weights before each trial. Random numbers are generated from [–0.3, 0.3] using the rand_uni function from Press et al. (1992). The order of the input data presentation is kept constant for each run to eliminate its effect on the result. The training process is stopped when $\kappa = 40,000$ consecutive iterations are unsuccessful.

Extensive computational experiments with different combinations of *H*- and δ-values have been performed on a DEC Alpha 375 Mhz. Table 2 summarises the results of the most important ones. Training performance is measured in terms of ARV(M_1) and validation performance in terms of ARV(M_2). The performance values represent the mean of 60 simulations, standard deviations are given in brackets. All simulations have similar computational complexity, so iterations to converge to the minimal ARV(M_2)-value may be used as a measure of learning time. It is easy to see that the combination of $H = 16$ and $\delta = 0.0025$ provides an appropriate choice for our particular application.

[14] This causes substantial differences in standard deviation of $t_{i\bullet}$ between M_1, M_2 and M_3 (see Table 1).

Table 2 The modular product unit neural network to model origin-constrained spatial interactions: Choice of H and δ (T = 1,000; N = 10)

Parameter	*Iterations*	*ARV*(M_1)	*ARV*(M_2)
H = 4			
δ = 0.0025	40,782 ± 19,886	0.2217 (0.0134)	0.1490 (0.0135)
H = 8			
δ = 0.0025	43,905 ± 18,854	0.2215 (0.0124)	0.1490 (0.0099)
H = 12			
δ = 0.0025	38,905 ± 17,896	0.2239 (0.0118)	0.1475 (0.0074)
H = 16			
δ = 0.0005	52,702 ± 33,311	0.2483 (0.0296)	0.1663 (0.0295)
δ = 0.0010	59,321 ± 49,647	0.2368 (0.0244)	0.1585 (0.0263)
δ = 0.0025	45,754 ± 21,284	0.2212 (0.0087)	0.1473 (0.0054)
δ = 0.0050	22,948 ± 15,360	0.2216 (0.0107)	0.1512 (0.0090)
δ = 0.0075	17,427 ± 12,918	0.2206 (0.0115)	0.1547 (0.0094)
δ = 0.0100	13,545 ± 11,753	0.2241 (0.0151)	0.1593 (0.0131)
H = 24			
δ = 0.0025	40,580 ± 20,047	0.2230 (0.0097)	0.1481 (0.0053)

Notes: ARV-performance values represent the mean (standard deviation in parentheses) of 60 simulations differing in the initial parameter values randomly chosen from [–0.3; 0.3]. Iterations: Number of iterations required to reach the parameter vector that provides the best ARV($M2$) performance. ARV($M1$): In-sample performance measured in terms of relative average variances. ARV($M2$): Validation performance measured in terms of relative average variances. M consists of 992 patterns, $M1$ of 496 patterns, and $M2$ of 248 patterns.

Figure 3 shows the learning curves of a typical run of the model (H = 16; measured by Alopex with T = 1,000; N = 10; δ = 0.0025) of the model in terms of ARV(M_1), ARV(M_2) and ARV(M_3) respectively. The term learning curve is used to characterise the performance as a function of iterations of the Alopex procedure. Figure 3(a) plots the ARV performance on the training set, Figure 3(b) the ARV performance on the validation set, and Figure 3(c) the ARV performance on the testing set.

Typically, the validation error oscillates rapidly at the beginning of the training process. Later the training process stabilises at around 5,000 iterations and the changes in the validation error become smaller. Instead of a clear increasing trend in the validation error that characterises overfitting, it starts to wander around some constant value at about 12,500. These undulations are caused by an increase of T in order to escape from shallow, local minima of the error surface, see Figure 3(d). Later, the training process stabilises and the changes in validation error become smaller. According to our termination criterion training is stopped after 18,841 iterations. At this stopping point P the model is used for testing (prediction).

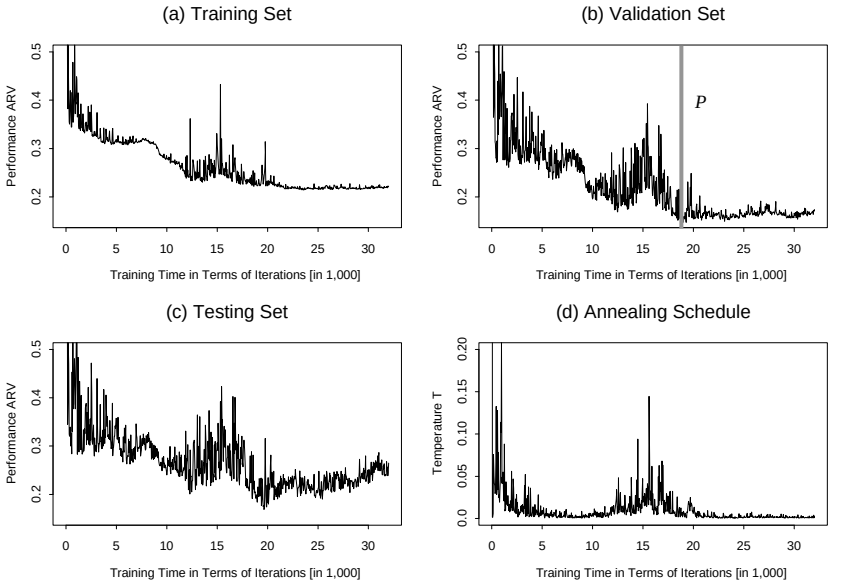


Figure 3 Training, validation and testing set curves as a function of training time (the vertical line *P* indicates the stopping point): The modular product unit neural network for modelling origin-constrained spatial interactions

5.4 The Benchmark Models

The first benchmark model is the standard origin-constrained gravity model, a special case of the generic interaction model of the gravity type, see Equation (5)[15]

$$\tau_{ij}^{grav} = b_{(i)}\, s_j^{\alpha}\, d_{ij}^{-\beta} \tag{12}$$

with

$$b_{(i)} = \frac{t_{i\bullet}}{\sum_{j \neq i} s_j^{\alpha}\, d_{ij}^{-\beta}} \tag{13}$$

where $b_{(i)}$ is the origin-specific balancing factor, α reflects the relationship of s_j with τ_{ij}^{grav}. β is the distance sensitivity parameter, $\beta > 0$. s_j is measured in terms of the

[15] There is virtual unanimity of opinion that site specific variables, such as s_j in this case, are generally best represented as power functions. The specification of f_{ij} is consistent with the general consensus that the power function is more appropriate for analysing longer distance interactions (Fotheringham and O'Kelly 1989).

gross regional product in j, d_{ij} is measured in terms of distances from i to j, while $t_{i\bullet}$ is measured in terms of erlang. We use the Alopex procedure for estimation with T = 1,000; N = 10, δ = 0.0075, and the termination criterion $\kappa = 40,000$ iterations.

The two-stage neural network modelling approach serves as second benchmark model. In the first stage the classical unconstrained neural spatial interaction model Ω_L is used, see Equation (7). The input data are preprocessed to logarithmically transformed data scaled to [0.1, 0.9]. The number of hidden summation units is 16. Least squares learning and the Alopex procedure ($T = 1,000$; $N = 10$; $\delta = 0.001$ and $\kappa = 40,000$) are used to determine the 81 model parameters. The parameters are randomly initialised in the range of [–0.3, 0.3]. In the second stage the following constraint mechanism is used to obtain origin-constrained flows

$$\Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})_{ij} = \frac{\Omega_L(\boldsymbol{x},\boldsymbol{w})_{ij}}{\sum_j \Omega_L(\boldsymbol{x},\boldsymbol{w})_{ij}} t_{i\bullet}. \tag{14}$$

5.5 Performance Tests and Results

Table 3 summarises the simulation results for the modular product unit network model ${}^{\pi}\Omega_{orig}$ in comparison with the two-stage neural network approach Ω_L^{constr} and the origin-constrained gravity model τ_{ij}^{grav}. Out-of-sample performance is measured in terms of ARV(M_3) and SRMSE(M_3). In addition, training performance values are displayed for matters of completeness. The figures represent averages taken over 60 simulations differing in the initial parameter values randomly chosen from [–0.3, 0.3]. Note that this random initialisation puts ${}^{\pi}\Omega_{orig}$ in contrast to Ω_L^{constr} at a slight disadvantage as its optimal range is [–2, +2].

Table 3 Benchmark comparisons of the modular product unit neural network ${}^{\pi}\Omega_{orig}$ with the two-stage neural network approach Ω_L^{constr} and the gravity model τ^{grav} for modelling origin-constrained spatial interactions

	Modular product unit neural network	*Two-stage neural network approach*	*Origin-constrained gravity model*
In-sample (training) performance			
ARV	0.2212 (0.0087)	0.2682 (0.0222)	0.2121 (0.0017)
SRMSE	1.2858 (0.0254)	1.4150 (0.0578)	1.2594 (0.0049)
Out-of-sample (testing) performance			
ARV	0.2022 (0.0150)	0.2251 (0.0255)	0.2262 (0.0027)
SRMSE	1.1017 (0.0407)	1.1614 (0.0670)	1.1658 (0.0069)

Notes: Figures represent averages taken over 60 simulations differing in the initial parameter values randomly chosen from [–0.3, 0.3] (standard deviation in parentheses); the training set consists of 496 patterns and the testing set of 248 patterns.

If out-of-sample (generalisation) performance is more important than fast learning, then the modular product unit neural network exhibits clear superiority. As can be seen by comparing the ARV-values and the SRMSE-values the modular product unit neural network model ranks best, followed by the two-stage neural network approach and the gravity model. The average generalisation performance, measured in terms of ARV(M_3), is 0.2022 and, measured in terms of SRMSE (M_3), 1.1017, compared to 0.2251 and 1.1614 in the case of the two-stage approach, and 0.2262 and 0.1658 in the case of the gravity model.[16] These differences in performance between the modular product unit neural network model and the benchmark models are statistically significant.[17]

However, if the goal is to minimise execution time and a sacrifice in generalisation accuracy is acceptable, then the standard origin-constrained gravity model is the model of choice. The gravity model outperforms the neural network models in terms of execution time, the modular product unit network model by a factor of 10 and the 2-stage neural network model by a factor of 10^2. However, note that this is essentially caused by two factors: first, that our implementations were done on a serial platform even though the neural network models are parallelisable, and, second, that we implemented a rather time consuming termination criterion (κ = 40,000) to stop the training process.

Residual Analysis. One means of further investigating the predictive power of the modular product unit neural network model ${}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})$ in comparison to the two benchmark models $\Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})$ and τ^{grav} is the use of residual analysis. Figure 4 displays this in terms of (i) the absolute residuals of the individual flows $[t_{ij} - {}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})]$ compared to $[t_{ij} - \Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})]$ and $[t_{ij} - \tau_{ij}^{grav}(\boldsymbol{x},\boldsymbol{w})]$, and the relative residuals of the individual flows $[t_{ij} - {}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})]/t_{ij}$ compared to $[t_{ij} - \Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})]/t_{ij}$ and $[t_{ij} - \tau_{ij}^{grav}(\boldsymbol{x},\boldsymbol{w})]/t_{ij}$, where both absolute and relative residuals are ordered by the size of the observed flows t_{ij}.

The main conclusions from this analysis can be summarised as follows: First, all three models show a tendency to underpredict larger flows. The neural network models underpredict 17 out of 25 flows in the largest decile, compared to 16 gravity model underpredictions. The benchmark models tend to give results with a little larger absolute deviation. Second, all three models show a tendency to overpredict smaller flows. This is evidenced by 23 overpredictions in the smallest decile, obtained by the modular product unit network, compared to 24 overpredictions of the benchmark models. Third, the modular product unit neural network model and the benchmark models show a relatively similar pattern of residuals.

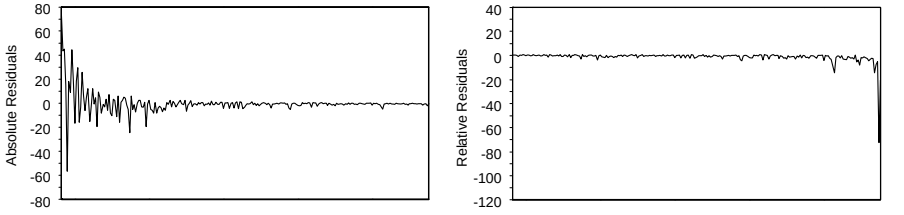
---

[16] ARV-out-of-sample variance of ${}^{\pi}\Omega_{orig}$ varies between 0.1725 and 0.2361, that of Ω_L^{constr} between 0.1852 and 0.2502 and that of τ^{grav} between 0.2225 and 0.2327. Note that ARV(M_3) and SRMSE (M_3) is smaller than ARV(M_1) and SRMSE(M_1). This might appear unusual at a first glance, but is due to the split of the data.

[17] The ARV- and SRMSE-differences are statistically significant at the one percent significance level: $\tilde{U}$ = 280, significance 0.0 compared to the standard origin-constrained gravity model and $\tilde{U}$ = 144, significance 0.009 compared to the two-stage approach.

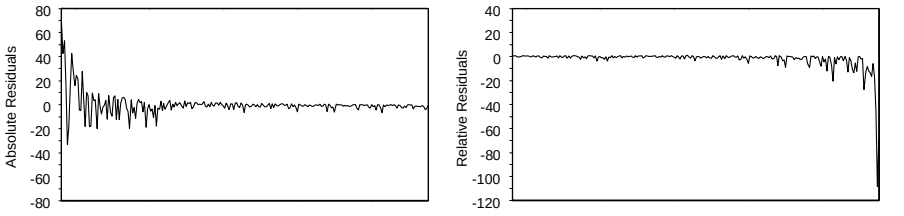
Despite this similarity the modular model tends to produce slightly more accurate predictions in the case of larger flows, but slightly less accurate ones in the case of smaller flows.

(i) Residuals of the modular product unit neural network model ${}^{\pi}\Omega_{orig}(\boldsymbol{x},\boldsymbol{w})$



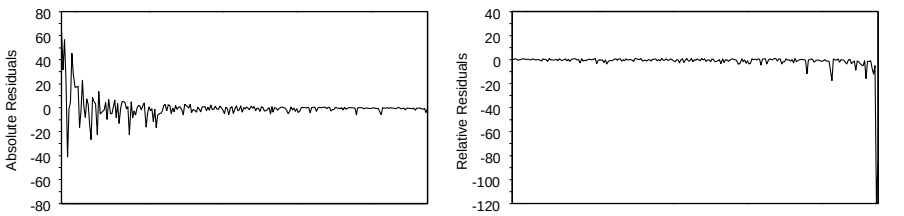
(ii1) Absolute Residuals $\left[t_{ij} - \Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})\right]$; ordered by the size of t_{ij} (ii2) Relative Residuals $\left[t_{ij} - \Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})\right]/\, t_{ij}$; ordered by the size of t_{ij}

(ii) Residuals of the two-stage neural network model approach $\Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})$



(ii1) Absolute Residuals $\left[t_{ij} - \Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})\right]$; ordered by the size of t_{ij} (ii2) Relative Residuals $\left[t_{ij} - \Omega_L^{constr}(\boldsymbol{x},\boldsymbol{w})\right]/\, t_{ij}$; ordered by the size of t_{ij}

(iii) Residuals of the origin-constrained gravity model τ^{grav}



(iii1) Absolute Residuals $\left[t_{ij} - \tau_{ij}^{grav}(\boldsymbol{x},\boldsymbol{w})\right]$; ordered by the size of t_{ij} (iii2) Relative Residuals $\left[t_{ij} - \tau_{ij}^{grav}(\boldsymbol{x},\boldsymbol{w})\right]/\, t_{ij}$; ordered by the size of t_{ij}

Figure 4 Residuals of the modular product unit neural network ${}^{\pi}\Omega_{orig}$, the two-stage neural network approach Ω_L^{constr} and the origin-constrained gravity model τ^{grav}

In summary, the analysis unequivocally shows that the modular product unit network outperforms the benchmark models in terms of both the ARV(M_3) and SRMSE(M_3) prediction performance, as well as in terms of the prediction accuracy, but the latter to a lesser degree than previously expected. One reason for

this might be that the method of stopped training combined with static data splitting to determine H and δ. This is an issue for further research (see Fischer and Reismann 2002).

6 Conclusions and Outlook

In this contribution, a novel neural network approach for modelling singly constrained spatial interactions was presented. The proposed function approximator is based on a modular network design with functionally independent product unit network modules where modularity refers to a decomposition on the computational level. Each module is a feedforward network with two inputs and a hidden layer of 16 product units and terminates with a single summation unit. The collective outputs of these modules constitute the input to the layer of output units that perform the flow prediction. The paper also demonstrates a simple way to implement the conservation rule from the viewpoint of origins (destinations) that avoids the need to modify the parameter estimation procedure to integrate the constraints on the predicted flows.

The Alopex procedure provides a powerful optimisation scheme that allows to produce LS-estimates of the model parameters. The dynamic interaction among a stochastic exploration process, the correlation based guidance to move towards regions of higher-quality solutions in the parameter space and the convergence-inducing process is responsible for the attractivity of the global search process of the procedure.

The attraction of this novel model approach depends not only on the awareness of what it can offer, but also on empirical illustrations of what can be gained in terms of out-of-sample (testing) approximation accuracy. Benchmark comparisons against the standard origin-constrained gravity model and the two-stage neural network approach, suggested by Openshaw (1998) and implemented by Mozolin et al. (2000), illustrate the superiority of the product unit neural network model, measured in terms of both the ARV- and the SRMSE-performance over 60 simulations.

The importance of avoiding overfitting cannot be overemphasised if a good predictive model is desired, and consequently, we believe that testing further techniques to control the model complexity without comprising network generalisation or learning accuracy is merited. In addition, research is needed in two further directions in future: first, in overcoming the shortcomings of least squares and normality assumptions in neural spatial interaction modelling that ignore the true integer nature of the flows; and second, in tackling the problem of sensitivity to the specific splitting of the data in parameter estimation and model testing (the so-called issue of fixed data splitting, see Fischer and Reismann 2002b and Fischer 2002).

References

Alonso W. (1978): A theory of movement. In: Hansen N.M. (ed.) *Human Settlement Systems*, Ballinger, Cambridge [MA], pp. 197-212

Aufhauser E. and Fischer M.M. (1985): Log-linear modelling and spatial analysis, *Environment and Planning A* 17 (7), 931-951

Bergkvist E. (2000): Forecasting interregional freight flows by gravity models, *Jahrbuch für Regionalwissenschaft* 20, 133-148

Bia A. (2000): A study of possible improvements to the Alopex training algorithm. In: *Proceedings of the VIth Brazilian Symposium on Neural Networks*, IEEE Computer Society Press, pp. 125-130

Bishop C.M. (1995): *Neural Networks for Pattern Recognition*, Oxford, Clarendon Press

Black W.R. (1995): Spatial interaction modelling using artificial neural networks, *Journal of Transport Geography* 3 (3), 159-166

Cover T.M. (1965): Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, *IEEE Transactions on Electronic Computers* 14 (3), 326-334

Durbin R. and Rumelhart D.E. (1989): Product units: A computationally powerful and biologically plausible extension to backpropagation, *Neural Computation* 1, 133-142

Fischer M.M. (2002): Learning in neural spatial interaction models: A statistical perspective, *Journal of Geographical Systems* 4(3), 287-299

Fischer M.M. (2000): Methodological challenges in neural spatial interaction modelling: The issue of model selection. In: Reggiani A. (ed.) *Spatial Economic Science: New Frontiers in Theory and Methodology*, Springer, Berlin, Heidelberg, New York, pp. 89-101

Fischer M.M. and Getis A. (1999): Introduction. New advances in spatial interaction theory, *Papers in Regional Science* 78 (2), 117-118

Fischer M.M. and Gopal S. (1994): Artificial neural networks: A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34 (4), 503-527

Fischer M.M. and Reismann M. (2002a): Evaluating neural spatial interaction modelling by bootstrapping, Paper presented at the 6th World Congress of the Regional Science Association International, Lugano, Switzerland, May 16-20, 2000 [accepted for publication in *Networks and Spatial Economics*].

Fischer M.M. and Reismann M. (2002b): A methodology for neural spatial interaction modelling, *Geographical Analysis* 34, 207-228

Fischer M.M., Hlavácková-Schindler K. and Reismann M. (1999): A global search procedure for parameter estimation in neural spatial interaction modelling, *Papers in Regional Science* 78 (2), 119-134

Fotheringham A.S. and O'Kelly M.E. (1989): *Spatial Interaction Models*: *Formulations and Applications*, Kluwer Academic Publishers, Dordrecht, Boston, London

Giles C. and Maxwell T. (1987): Learning, invariance, and generalization in high-order neural networks, *Applied Optics* 26 (23), 4972-4978

Gopal S. and Fischer M.M. (1993): Neural network based interregional telephone traffic models. In: *Proceedings of the International Joint Conference on Neural Networks IJCNN 93 Nagoya*, Japan, October 25-29, pp. 2041-2044

Harth E. and Pandya A.S. (1988): Dynamics of ALOPEX process: Application to optimization problems. In: Ricciardi L.M. (ed.) *Biomathematics and Related Computational Problems*. Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 459-471

Hassoun M.H. (1995): *Fundamentals of Neural Networks*, MIT Press, Cambridge [MA] and London [England]

Hecht-Nielsen R. (1990): *Neurocomputing*, Addison-Wesley, Reading [MA]

Hornik K., Stinchcombe M. and White H. (1989): Multi-layer feedforward networks are universal approximators, *Neural Networks* 2 (5), 359-366

Mozolin M., Thill J.-C. and Usery E.L. (2000): Trip distribution forecasting with multilayer perceptron neural networks: A critical evaluation, *Transportation Research B* 34 (1), 53-73

Openshaw S. (1998): Neural network, genetic, and fuzzy logic models of spatial interaction, *Environment and Planning A* 30 (11), 1857-1872

Openshaw S. (1993): Modelling spatial interaction using a neural net. In: Fischer M.M. and Nijkamp P. (eds.) *Geographic Information Systems, Spatial Modelling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 147-164

Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P. (1992): *Numerical Recipes in C*: *The Art of Scientific Computing*, Cambridge University Press, Cambridge [MA]

Reggiani A. and Tritapepe T. (2000): Neural networks and logit models applied to commuters' mobility in the metropolitan area of milan. In: Himanen V., Nijkamp P. and Reggiani A. (eds.) *Neural Networks in Transport Applications*, Ashgate, Aldershot, pp. 111-129

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

Sen A. and Smith T.E. (1995): *Gravity Models of Spatial Interaction Behavior*, Springer Berlin, Heidelberg, New York

Senior M.L. (1979): From gravity modelling to entropy maximizing: A pedagogic guide, *Progress in Human Geography* 3 (2), 175-210

Tobler W. (1983): An alternative formulation for spatial interaction modelling, *Environment and Planning A* 15 (5), 693-703

Turton I., Openshaw S. and Diplock G.J. (1997): A genetic programming approach to building new spatial models relevant to GIS. In: Kemp Z. (ed.) *Innovations in GIS 4*, Taylor & Francis, London, pp. 89-104

Unnikrishnan K.P. and Venugopal K.P. (1994): Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks, *Neural Computation* 6 (3), 469-490

Weigend A.S., David E.R. and Huberman B.A. (1991): Back-propagation, weight-elimination and time series prediction. In: Touretzky D.S., Elman J.L., Sejnowski T.J. and Hinton G.E. (eds.) *Connectionist Models: Proceedings of the 1990 Summer School*, Morgan Kaufmann Publishers, San Mateo [CA], pp. 105-116

White H. (1980): Using least squares to approximate unknown regression functions, *International Economic Review* 21 (1), 149-170

Wilson A.G. (1967): A statistical theory of spatial distribution models, *Transportation Research* 1, 253-269

13 Learning in Neural Spatial Interaction Models: A Statistical Perspective

This chapter views learning as an unconstrained nonlinear minimisation problem in which the objective function is defined by the negative log-likelihood function and the search space by the parameter space of an origin-constrained product unit neural spatial interaction model. We consider Alopex based global search, as opposed to local search based upon backpropagation of gradient descents, each in combination with the bootstrapping pairs approach to solve the maximum likelihood learning problem. Interregional telecommunication traffic flow data from Austria are used as test bed for comparing the performance of the two learning procedures. The study illustrates the superiority of Alopex based global search, measured in terms of Kullback and Leibler's information criterion.

1 Introduction

In many spatial interaction contexts, little is known about the form of the spatial interaction function that is to be approximated. In such cases it is not possible to utilise a parametric modelling approach where a mathematical model is specified with unknown coefficients that have to be estimated. Neural spatial interaction models relieve the model user of the need to specify exactly a model that includes all necessary terms to model the true spatial interaction function. Two major issues have to be solved when applying a neural spatial interaction model in a real world context: first the representation problem, and, second the learning problem. Our interest centers at the latter problem.

This contribution departs from earlier studies in neural spatial interaction modelling in three respects. *First*, current research generally suffers from least squares and Gaussian assumptions that ignore the true integer nature of the flows and approximate a discrete-valued process by an almost certainly misrepresentative distribution. To overcome this deficiency we adopt a more suitable approach for solving the learning problem, namely maximum likelihood learning (estimation) under more realistic distributional assumptions of Poisson processes. *Second*, classical (i.e. unconstrained summation unit) neural spatial interaction models represent – no doubt – a rich and flexible class of spatial interaction function approximators to predict flows, but may be of little practical value if a priori information is available on accounting constraints on the predicted flows. We focus attention on the only existing generic neural network model for the case of spatial

interaction. *Third*, we utilise the bootstrapping pairs approach with replacement to overcome the generally neglected issue of fixed data splitting and to get a better statistical picture of the learning and generalisation variability of the model concerned.

Succinctly put, the objective of the paper is twofold. *First*, we develop a rationale for specifying the maximum likelihood learning problem in product unit neural networks for modelling origin-constrained spatial interaction flows as recently introduced in Fischer et al. (2003). *Second*, we consider Alopex based global search, and local search based upon backpropagation of gradient descents, in combination with the bootstrapping pairs approach to solve the maximum likelihood learning problem.

The paper proceeds as follows. The next section sets forth the context in which the learning problem is considered. Section 3 views learning as an unconstrained nonlinear minimisation problem in which the objective function is defined by the negative log-likelihood and the search space by the parameter space. In the sections that follow we discuss details how the highly nonlinear learning problem can be solved. We consider two learning procedures in some more detail: gradient descent based local search (the most widely used technique in *unconstrained* neural spatial interaction modeling) in Section 4 and Alopex based global search in Section 5. Section 6 serves to illustrate the application of these procedures in combination with the bootstrapping pairs approach to address the issue of network learning. Interregional telecommunication traffic flow data are utilised as test bed for evaluating the two competing learning approaches. The robustness of the procedures is measured in terms of Kullback and Leibler's information criterion. Section 7 outlines some directions for future research.

2 The Context

Before discussing the learning problem we must specify the context in which we consider learning. Our attention is focused on learning in origin-constrained product unit neural spatial interaction models. Throughout the paper we will be concerned with the data generated according to the following conditions.

Assumption A: Observed data are the realisation of the sequence $\{Z_u = (X_u, Y_u), u = 1, ..., U\}$ of $(N+1)\times 1$ independent vectors $(N \in \mathbb{N})$ defined as a Poisson probability space.

The random variables Y_u represent targets. Their relationship to the variables X_u is of primary interest. When $E(Y_u) < \infty$, the conditional expectation of Y_u given X_u exists, denoted as $g = E(Y_u \mid X_u)$. Defining $\varepsilon_u \equiv Y_u - g(X_u)$

$$Y_u = g(X_u) + \varepsilon_u \tag{1}$$

The unknown spatial interaction function g embodies the systematic part of the stochastic relation between Y_u and X_u. The error ε_u is noise, with the property

$E(\varepsilon_u \mid X_u) = 0$ by construction. Our problem is to learn the mapping g from a realisation of the sequence $\{Z_u\}$.

We are interested in learning the mapping g for the case of origin-constrained spatial interaction. Because g is unknown, we approximate it using a family of known functions. Of particular interest to us are the output functions of origin-constrained product unit neural spatial interaction models as recently introduced in Fischer et al. (2003).

Assumption B: Model output is given by

$$\Omega^H(\mathbf{x},\mathbf{w})_j = \psi_j\left(\sum_{h=1}^{H}\gamma_h\ \varphi_h\left(\prod_{n=2j-1}^{2j} x_n^{\beta_{hn}}\right)\right) \tag{2}$$

for $j = 1,...,J$ with $\varphi_h, \psi_h : \Re \to \Re$, and $x \in \Re^{2J}$, that is $\mathbf{x} = (x_1, x_2, \ldots, x_{2j\text{-}1}, ..., x_{2J\text{-}1}, x_{2J})$ where x_{2j-1} represents a variable pertaining to destination j ($j = 1, ..., J$) and x_{2j} a variable characterising the separation from region i to region j ($i = 1,..., I$; $j = 1,..., J$) of the spatial interaction system under scrutiny. β_{hn} ($h = 1, ..., H$; $n = 2j-1, 2j$) are the input-to-hidden connection weights, and γ_h ($h = 1, ..., H$) the hidden-to-output weights in the jth module of the network model. The symbol $\mathbf{w}$ is a convenient shorthand notation of the $(3H)$-dimensional vector of all the model parameters. ψ_j ($j = 1, ..., J$) represents a nonlinear summation unit and φ_h ($h = 1, ..., H$) a linear hidden product unit transfer function. The model output function is explicitly indexed by the number, H, of hidden units in order to indicate the dependence. Finally, it is worth noting that models of type (2) utilise a product unit rather than the generally used standard summation unit neural network framework for modelling interactions over space. The product units compute the product of inputs, each raised to a variable power.

A leading case that is considered in this paper occurs when $\varphi_h(\cdot)$ is specified as identity function and $\psi_j(\cdot)$ as a nonlinear transfer function which resembles the Bradley-Terry-Luce model augmented by a bias unit $\tilde{b}_{(i)}$ to build the a priori information into the model structure (for a mathematical derivation see Fischer et al. 2003):

$$\Omega^H(\mathbf{x},\mathbf{w})_j = \tilde{b}_{(i)}\frac{\sum_{h=1}^{H}\gamma_h\prod_{n=2j-1}^{2j}x_n^{\beta_{hn}}}{\sum_{j'=1}^{J}\sum_{h'=1}^{H}\gamma_{h'}\prod_{n=2j'-1}^{2j'}x_n^{\beta_{h'n}}} \qquad j = 1, ..., J \tag{3}$$

for $j = 1, ..., J$. $\tilde{b}_{(i)}$ is the bias signal generated by a dummy unit whose output is clamped at the scalar $t_{i\bullet}$, where $t_{i\bullet}$ denotes the observed flow from region i to each of the J regions.

3 The Learning Problem

If we view (3) as generating a family of approximations – as $\boldsymbol{w}$ ranges over $\boldsymbol{W}$, say – to a spatial interaction function g, then we need a way to pick the best approximation from this family. This is the function of network learning (also termed training or parameter estimation). It is convenient to consider learning as an unconstrained nonlinear minimisation problem in which the objective function is defined by a loss (error, cost) function and the search space by the (3H)-dimensional parameter space. Formally,

$$\min_{\boldsymbol{w} \in \boldsymbol{W}} \lambda(\boldsymbol{w}) \tag{4}$$

where $\lambda(\boldsymbol{w})$ represents the loss function measuring the network performance given the parameter $\boldsymbol{w}$ and observation $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$. It is evident that the choice of the loss function plays a crucial role in the determination of the optimal parameter $\hat{\boldsymbol{w}}$. We follow Fischer and Reismann (2002b) to specify an appropriate loss function. Hereby, we assume that the objective is to find that neural spatial interaction model which is the most likely explanation of the observed data set (Rumelhart et al. 1995). We express this as attempting to maximise

$$P(\Omega^H(\boldsymbol{w}) \mid M) = \frac{P(M \mid \Omega^H(\boldsymbol{w}))\ P(\Omega^H(\boldsymbol{w}))}{P(M)} \tag{5}$$

where $P(M \mid \Omega^H(\boldsymbol{w}))$ is the probability that model $\Omega^H(\boldsymbol{w})$ would have produced the observed data M. $P(\Omega^H(\boldsymbol{w}))$ represents the unconditional probability density of $\Omega^H(\boldsymbol{w})$ and $P(M)$ that of M.

Since sums are easier to work with than products, we will maximise the log of $P(\Omega^H(\boldsymbol{w}) \mid M)$, and since this log is a monotonic transformation, maximising the log is equivalent to maximising the probability itself. In this case we get

$$\ln P(\Omega^H(\boldsymbol{w}) \mid M) = \ln P(M \mid \Omega^H(\boldsymbol{w})) + \ln P(\Omega^H(\boldsymbol{w})) - \ln P(M). \tag{6}$$

The probability $P(M)$ of the data is not dependent on $\Omega^H(\boldsymbol{w})$. Thus, it is sufficient to maximise the first two terms of the right hand side of Equation (6). The first of these terms represents the probability of the data given the model, and hence measures how well the network accounts for the data. The second term is a representation of the model itself; that is, it is a prior probability of the model that can be utilised to get information and constraints into the learning procedure.

We focus solely on the first term, the performance, and begin by noting that the data can be broken down into a set of observations, $M = \{z_u = (x_u, y_u) \mid u = 1; \ldots, U\}$, each z_u, we will assume chosen independently of the others. Hence we can write the probability of the data given the model as

$$\ln P(M \mid \Omega^H(\boldsymbol{w})) = \ln \prod_u P(z_u \mid \Omega^H(\boldsymbol{w})) = \sum_u \ln P(z_u \mid \Omega^H(\boldsymbol{w})). \tag{7}$$

Note that this assumption permits to express the probability of the data given the model as the sum of terms, each term representing the probability of a single observation given the model. We can still take another step and break the data into two parts: the observed input data x_u and the observed target data y_u. Therefore we can write

$$\ln P(M \mid \Omega^H(\boldsymbol{w})) = \sum_u \ln P(y_u \mid x_u \text{ and } \Omega^H(\boldsymbol{w})_u) + \sum_u \ln P(x_u). \tag{8}$$

Since we assume that x_u does not depend on the model, the second term of Equation (8) will not affect the determination of the optimal model. Thus, we need only to maximise the first term of the right-hand side of Equation (8).

Up to now we have – in effect – made only the assumption of the independence of the observed data. To proceed further, we have to specify the form of the distribution of which the model output is the mean. In line with *Assumption A* that the observed data are the realisation of a sequence of independent Poisson random variables we can write the probability of the data given the model as

$$P(y_u \mid x_u \text{ and } \Omega^H(\boldsymbol{w}))_u = \frac{\prod_u \Omega^H(\boldsymbol{w})_u^{y_u} \exp(-\Omega^H(\boldsymbol{w})_u)}{y_u!} \tag{9}$$

and, hence, define a maximum likelihood estimator as the parameter that maximises the log-likelihood function

$$\max_{\boldsymbol{w} \in \boldsymbol{W}} L(\boldsymbol{w}) = \max_{\boldsymbol{w} \in \boldsymbol{W}} \sum_u (y_u \ln \Omega^H(\boldsymbol{w})_u - \Omega^H(\boldsymbol{w})_u). \tag{10}$$

Instead of maximising the log-likelihood it is more convenient to minimise the negative log-likelihood function $\lambda(\boldsymbol{w})$

$$\min_{\boldsymbol{w} \in \boldsymbol{W}} \lambda(\boldsymbol{w}) = \min_{\boldsymbol{w} \in \boldsymbol{W}} \left[-\sum_u y_u \ln \Omega^H(\boldsymbol{w})_u - \Omega^H(\boldsymbol{w})_u \right]. \tag{11}$$

The function λ is called the loss, cost or objective function. $\boldsymbol{w}$ is a (3*H*)-dimensional vector called the design vector. The point $\hat{\boldsymbol{w}}$ is a *global minimiser* for $\lambda(\boldsymbol{w})$ if $\lambda(\hat{\boldsymbol{w}}) \le \lambda(\boldsymbol{w})$ for all $\boldsymbol{w} \in \Re^{3H}$. A parameter vector $\hat{\boldsymbol{w}}$ is a *strict local minimiser* for $\lambda(\boldsymbol{w})$ if the relation $\lambda(\hat{\boldsymbol{w}}) \le \lambda(\boldsymbol{w})$ holds for a ball $B(\hat{\boldsymbol{w}}, \in)$. If the first and second derivatives of $\lambda(\boldsymbol{w})$, a point $\hat{\boldsymbol{w}}$ is a strict local minimiser of $\lambda(\boldsymbol{w})$ if the gradient is zero [that is $\nabla\lambda(\hat{\boldsymbol{w}}) = 0$] and the Hessian matrix is positive definite [that is, $\boldsymbol{w}^{\mathrm{T}} \nabla^2 \lambda(\hat{\boldsymbol{w}}) > 0$]. λ is typically a highly nonlinear function

of the parameters. As a consequence, it is in general not possible to find closed-form solutions for the minima.

In the sections that follow we discuss how the learning problem (11) can be solved. We seek a solution to what is typically a highly nonlinear optimisation problem. We first consider the gradient descent based search and then the Alopex based global search procedures.

4 Gradient Descent Based Search

The most prominent procedures solving the learning problem (11) are gradient descent techniques. These methods transform the minimisation problem into an associated system of first-order ordinary differential equations which can be written in compact matrix form (see Cichocki and Unbehauen 1993) as

$$\frac{d\boldsymbol{w}}{ds} = -\mu(\boldsymbol{w}, s)\ \nabla_w \lambda(\boldsymbol{w}) \tag{12}$$

with

$$\frac{d\boldsymbol{w}}{ds} = \left[\frac{dw_1}{ds}, ..., \frac{dw_{3H}}{ds}\right]^{\mathrm{T}}. \tag{13}$$

$\nabla_w \lambda(\boldsymbol{w})$ represents the gradient operator of $\lambda(\boldsymbol{w})$ with respect to the $(3H)$-dimensional parameter vector $\boldsymbol{w}$. $\mu(\boldsymbol{w}, s)$ denotes a $3H \times 3H$ positive definite symmetric matrix with entries depending on time s and the vector $\boldsymbol{w}(s)$.

In order to find the desired vector $\hat{\boldsymbol{w}}$ that minimises the loss function $\lambda(\boldsymbol{w})$ we need to solve the system of ordinary equations (12) with initial conditions. Thus, the minima of $\lambda(\boldsymbol{w})$ are determined by the following trajectory of the gradient system with

$$\hat{\boldsymbol{w}} = \lim_{s \to \infty} \boldsymbol{w}(s). \tag{14}$$

But it is important to note that we are concerned only with finding the limit rather that determining a detailed picture of the whole trajectory $\boldsymbol{w}(s)$ itself. In order to illustrate that the system of differential equations given by (12) is stable let us determine the time derivative of the loss function

$$\frac{d\lambda}{ds} = \sum_{k=1}^{3H} \frac{\partial \lambda}{\partial w_k} \frac{\partial w_k}{\partial s} = -\left[\nabla_w \lambda(\boldsymbol{w})\right]^{\mathrm{T}} \mu(\boldsymbol{w}, s) \nabla_w \lambda(\boldsymbol{w}) \leq 0 \tag{15}$$

under the condition that the matrix $\mu(\boldsymbol{w},s)$ is symmetric and positive definite. Relation (15) guarantees under appropriate regularity conditions that the loss function decreases in time and converges to a stable local minimum as $s \to \infty$. When $d\boldsymbol{w}/ds = \mathbf{0}$ then this implies $\nabla\lambda(\boldsymbol{w}) = \mathbf{0}$ for the system of differential equations. Thus, the stable point coincides either with the minimum or with the inflection point of the loss function (see Cichocki and Unbehauen 1993).

The speed of convergence to the minimum depends on the choice of the entries of $\mu(\boldsymbol{w},s)$. Different choices for μ implement different specific gradient based search procedures: In the simplest and most popular procedure, known as gradient descent, the matrix $\mu(\boldsymbol{w},s)$ is reduced to the unity matrix multiplied by a positive constant η that is called the learning parameter. It is interesting to note that the vectors $d\boldsymbol{w}/ds$ and $\nabla\lambda(\boldsymbol{w})$ are opposite vectors. Hence, the time evaluation of $\boldsymbol{w}(s)$ will result in the minimisation of $\lambda(\boldsymbol{w})$ as time s goes on. The trajectory $\boldsymbol{w}(s)$ moves along the direction which has the sharpest rate of decrease and is called the direction of steepest descent.

The discrete-time version of the steepest descent (also termed gradient) procedure can be written in vector form as

$$\boldsymbol{w}(s+1) \;=\; \boldsymbol{w}(s) - \eta(s)\;\nabla_{w}\;\lambda(\boldsymbol{w}(s)) \tag{16}$$

with $\eta(s) \geq 0$. The parameter $\eta(s)$ is called learning rate and determines the length of the step to be taken in the direction of the gradient of $\lambda(\boldsymbol{w}(s))$. It is important to note that $\eta(s)$ should be bounded in a small range to ensure stability of the algorithm. Note that the sometimes extreme local irregularity ('roughness', 'ruggedness') of the function λ over $\boldsymbol{W}$ arising in neural spatial interaction models may require the development and use of appropriate modifications of the standard procedure given by (16).

We utilise the simplest version of (16), that is, $\eta(s) = \eta$ (η sufficiently small) in combination with the technique of backpropagation popularised in a paper by Rumelhart et al. (1986) for evaluating the derivatives of the loss function with respect to the parameters. This technique provides a computationally efficient method for evaluating such derivatives. It corresponds to a propagation of errors backwards through the spatial interaction network. Because of the relative familiarity of this evaluation technique we do not go into details regarding the specifics of implementation. Those not familiar with backpropagation are referred to Bishop (1995) for further information.

5 Alopex Based Global Search

Although computationally efficient, gradient based minimisation procedures, such as backpropagation of gradient errors, may lead only to local minima of $\lambda(\boldsymbol{w})$ that happen to be close to the initial search point $\boldsymbol{w}(0)$. As a consequence, the quality

of the final solution of the learning problem is highly dependent on the selection of the initial condition. Global search procedures are expected to lead to optimal or 'near-optimal' parameter configurations by allowing the network model to escape from local minima during training. Genetic algorithms and the Alopex procedure are attractive candidates. We utilise the latter as described in Fischer and Reismann (2002b).

The success of global search procedures in finding a global minimum of a given function such as $\lambda(\boldsymbol{w})$ over $\boldsymbol{w} \in \boldsymbol{W}$ hinges on the balance between an exploration process, a guidance process and a convergence-inducing process (see Hassoun 1995). The *exploration process* gives the search a mechanism for sampling a sufficiently diverse set of parameters $\boldsymbol{w}$ in $\boldsymbol{W}$. The Alopex procedure performs an exploration process that is stochastic in nature. The *guidance process* is an implicit process that evaluates the relative quality of search points and utilises correlation guidance to move towards regions of higher quality solutions in the parameter space. Finally, the *convergence-inducing process* ensures the convergence of the search to find a fixed solution $\hat{\boldsymbol{w}}$. The convergence-inducing process is realised effectively by a parameter *T*, called temperature in analogy to the simulated annealing procedure, that is gradually decreased over time. The dynamic interaction among these three processes is responsible for giving the Alopex search procedure its global optimising character.

Alopex is a correlation-based method for solving the learning problem (see Bia 2000; Unnikrishnan and Venugopal 1994; Harth and Pandya 1988). The loss function λ is minimised by means of weight changes that are calculated for the *s*th step ($s > 2$) of the iteration process as follows:

$$w_k(s) \;=\; w_k(s-1) \;+\; \delta \;\operatorname{sgn}(\sigma - p_k(s)) \tag{17}$$

where δ is the step size that has to be chosen a priori, and σ is an uniformly distributed random value with $\sigma \in [0,1]$. The probability of change of the parameter is calculated as

$$p_k(s) \;=\; \left[1 + \exp\left\{-\frac{C_k(s)}{T(s)}\right\}\right]^{-1} \tag{18}$$

with $C_k(s)$ given by the correlation

$$C_k(s) \;=\; \big[w_k(s-1) - w_k(s-2)\big]\big[\lambda(w_k(s-1)) - \lambda(w_k(s-2))\big] \;=\; \Delta w_k(s)\,\Delta\lambda(w_k(s)). \tag{19}$$

The weight will be incremented in a given fixed magnitude δ, when $\Delta w_k(s) > 0$, and the opposite when it is less than zero. The sign of C_k indicates whether λ varies in the same way as w_k. If $C_k > 0$, both λ and w_k will be raised or lowered. If $C_k < 0$, one will be lowered and the other one raised. If *T* is too small,

the algorithm gets trapped into local minima of λ. Thus the value of T for each iteration, $T(s)$, is chosen using the following heuristic 'annealing schedule':

$$T(s) = \begin{cases} \frac{\delta}{3HS} \sum_{k} \sum_{s'=s-S}^{s-1} |C_k(s')| & \text{if } s \text{ is a multiple of } S \\ T(s-1) & \text{otherwise} \end{cases} \tag{20}$$

where $3H$ denotes the number of parameters. The annealing schedule controls the randomness of the algorithm. When T is small, the probability of changing the parameter is around zero if C_k is negative and around one if C_k is positive.

If T is large, then $p_k \cong 0.5$. This means that there is the same probability to increment or decrement the weights and that the direction of the steps is now random. In other words, high values of T imply a random walk, while low values cause a better correlation guidance (see Bia 2000). The effectiveness of Alopex in locating global minima and its speed of convergence critically depend on the balance of the size of the feedback term $\Delta w_k \ \Delta\lambda$ and the temperature T. If T is very large compared to $\Delta w_k \ \Delta\lambda$ the process does not converge. If T is too small, a premature convergence to a local minimum might occur.

The algorithm has three control parameters: the initial temperature T, the number of iterations S over which the correlations are averaged for annealing, and the step size δ. Setting the temperature high initially, say $T = 1,000$, one may escape from local minima. The temperature is lowered at an appropriate rate so as to control the probability of jumping away from relatively good minima. The correlations need to be averaged over a sufficiently large number of iterations so that the annealing does not freeze the algorithm at local minima. $S = 10$ has been found to be appropriate. δ is a critical control parameter that has to be chosen with care.

It is worth noting that Alopex based global search is similar to the method of simulated annealing (see Kirkpatrick et al. 1983), but differs in three important aspects: *first*, the correlation $(\Delta\lambda \ \Delta\boldsymbol{w})$ is used instead of the change in error $\Delta\lambda$ for parameter updates; *second*, all parameter changes are accepted at every iteration step; and *third* during an iteration all parameters are updated simultaneously.

6 Experimental Environment and Performance Tests

To analyse the performance of the learning procedures discussed in the previous sections in a real world context we utilise the interregional telecommunication traffic flow data from Austria as test bed.

6.1 The Data Set

The data set was constructed from three data sources: a $(32,32)$-interregional flow matrix (t_{ij}), a $(32,32)$-distance matrix (d_{ij}), and gross regional products q_j for the 32 telecommunication regions. It contains 992 3-tuples (q_i, d_{ij}, t_{ij}) where the first two components represent the input variables x_{2j-1} and x_{2j} of the jth module of the network model, and the last component the target output. Input data were preprocessed to lie in [0.1, 0.9]. The telecommunication data stem from network measurements of carried telecommunication traffic in Austria in 1991, in terms of erlang, which is defined as the number of phone calls (including facsimile transmissions) multiplied by the average length of the call (transfer) divided by the duration of the measurement.

6.2 Data Splitting, Bootstrapping and Performance Measure

The main goal of network learning is to minimise $\lambda(\boldsymbol{w})$ while ensuring good model generalisation. Thus, we monitor model performance during training to assure that further learning improves generalisation as well as reduces the loss function λ. For this purpose an additional set of internal validation data, independent from the training data, is used. In our implementation of the learning procedures network learning will be stopped when $\kappa = 40,000$ consecutive iterations are unsuccessful. κ has been chosen so large at the expense of the greater training time, to ensure more reliable estimates. Of course, setting the number of unsuccessful iterations to 40,000 (or more) does not guarantee that there would be any successful steps ahead if training continued. At some stage a learning algorithm may recover from some local attractor and accomplish further error minimisation, but we require it should occur within a certain number of iterations.

One of the simplest methods for assessing the learning and generalisation abilities of a model is, thus, data splitting. This method simulates learning and generalisation by partitioning the total data set $M = \{(x_u, y_u),\ u = 1, ..., U\}$ into three separate subsets: the training set $M_1 = \{(x_{u1}, y_{u1}),\ u1 = 1, ..., U_1\}$, the internal validation set $M_2 = \{(x_{u2}, y_{u2}),\ u2 = 1, ..., U_2\}$ and the test set $M_3 = \{(x_{u3}, y_{u3}),\ u3 = 1, ..., U_3\}$. M_1 is used for learning only, M_2 for stopping the learning process and M_3 for measuring the generalisation performance. In our study $U_1 = 496$, $U_2 = U_3 = 248$.

It is common practice to use random splits of the data. The simplicity of this approach is appealing. But recent experience has found this approach to be more sensitive to the specific splitting of the data (see Fischer and Reismann 2002a). In order to overcome this problem we use the learning algorithms in combination with the bootstrapping pairs approach with replacement $[B = 60]$ (see Efron 1982) to address the issue of network learning. This approach combines the purity of splitting the data into three disjoint data sets with the power of a resampling

procedure and, thus, allows to get a better statistical picture of both the learning and prediction variability.

Performance is measured in terms of Kullback and Leibler's information criterion (see Kullback and Leibler 1951), that reflects the conditions under which ML learning is to be evaluated

$$KLIC(M) = \sum_{u=1}^{U} \frac{y_u}{\sum_{u'=1}^{U} y_{u'}} \ln\left[\frac{y_u \left[\sum_{u'=1}^{U} y_{u'} \right]^{-1}}{\Omega^H(x_u, \boldsymbol{w}) \left[\sum_{u'=1}^{U} \Omega^H(x_{u'}, \boldsymbol{w}) \right]^{-1}} \right] \tag{21}$$

where (x_u, y_u) denotes the uth pattern of the data set M. The performance measure has a minimum at zero and a maximum at positive infinity when $y_u > 0$ and $\Omega^H(x_u, \boldsymbol{w}) = 0$ for any (x_u, y_u)-pair.

6.3 Performance Tests

Both methods, backpropagation of gradient descents and Alopex are iterative procedures. This implies that the learning process is more or less sensitive to its starting point in both cases. The solutions to the learning problem may vary as the initial parameter settings are changed. Despite recent progress in finding the most appropriate parameter initialisation to determine near optimal solutions, the most widely adopted approach still uses random parameter initialisation. In our experiments random numbers were generated from [–0.3, 0.3] using the rand_uni function from Press et al. (1992). The order of the input data presentation was kept constant for each run to eliminate its effect on the result.

For concreteness, we consider the learning problem in a series of increasingly complex neural spatial interaction models $\{\Omega^H(\boldsymbol{x}, \boldsymbol{w}), H = 2, 4, 6, 8, 10, 12, 14\}$ permitting the complexity of the product unit neural network to grow at an appropriate rate. Statistical theory may provide guidance in choosing the control parameters of the learning algorithms for optimal tracking, but this is a difficult area for future research. In this study the Alopex parameters T and S were set to 1,000 and 10, respectively. In order to do justice to each learning procedure, the critical Alopex control parameter δ (step size) and the critical gradient descent control parameter η (learning rate) were systematically sought for each Ω^H. Extensive computational experiments with $\eta \in \{0.0000025, 0.0000050, 0.0000100, 0.0000250, 0.0000500, 0.0001000, 0.0002500\}$ and $\delta \in \{0.0005, 0.0010, 0.0025, 0.0050, 0.0075, 0.0100, 0.0250, 0.0500, 0.1000\}$ have been performed on DEC Alpha 375 MHz to address the issue of learning in the above models.

Table 1 shows the best solutions of both procedures for Ω^H with H = 2, 4, 6, 8, 10, 12 and 14. Learning (in-sample) performance is measured in terms of $KLIC(M_1)$, validation performance in terms of $KLIC(M_2)$ and generalisation (out-of-sample) performance in terms of $KLIC(M_3)$. The performance values represent the mean of $B = 60$ bootstrap replications, standard deviations are given

in brackets. The results achieved illustrate that Alopex based global search outperforms backpropagation of gradient descents in all cases, in terms of both learning and generalisation performance. There is also strong evidence of the robustness of the algorithm, measured in terms of standard deviation. We attribute Alopex superiority in finding better local minima to its annealing mechanism to escape from local minima during training.

Table 1 Approximation to the spatial interaction function using backpropagation of gradient descents versus Alopex based global search

	Parameter	*Backpropagation of gradient descents*			*Parameter*	*Alopex based global search*		
		KLIC(M_1)	KLIC(M_2)	KLIC(M_3)		KLIC(M_1)	KLIC(M_2)	KLIC(M_3)
$H = 2$	$\eta = 10^{-5}$	0.2105 (0.0540)	0.2230 (0.0911)	0.2262 (0.0812)	$\delta = 10^{-2}$	0.1927 (0.0522)	0.1968 (0.0776)	0.2120 (0.0698)
$H = 4$	$\eta = 10^{-5}$	0.2109 (0.0541)	0.2229 (0.0909)	0.2262 (0.0806)	$\delta = 10^{-2}$	0.1853 (0.0460)	0.1897 (0.0754)	0.2035 (0.0690)
$H = 6$	$\eta = 10^{-5}$	0.2125 (0.0551)	0.2231 (0.0895)	0.2271 (0.0796)	$\delta = 2.5 \cdot 10^{-2}$	0.1883 (0.0483)	0.1902 (0.0725)	0.2048 (0.0708)
$H = 8$	$\eta = 10^{-5}$	0.2129 (0.0553)	0.2230 (0.0879)	0.2279 (0.0796)	$\delta = 2.5 \cdot 10^{-2}$	0.1868 (0.0505)	0.1888 (0.0732)	0.2049 (0.0707)
$H = 10$	$\eta = 5 \cdot 10^{-6}$	0.2120 (0.0543)	0.2243 (0.0887)	0.2273 (0.0811)	$\delta = 2.5 \cdot 10^{-2}$	0.1874 (0.0485)	0.1897 (0.0734)	0.2045 (0.0691)
$H = 12$	$\eta = 2.5 \cdot 10^{-5}$	0.2131 (0.0560)	0.2254 (0.0893)	0.2283 (0.0826)	$\delta = 10^{-2}$	0.1866 (0.0483)	0.1909 (0.0731)	0.2019 (0.0684)
$H = 14$	$\eta = 5 \cdot 10^{-6}$	0.2122 (0.0547)	0.2260 (0.0894)	0.2275 (0.0803)	$\delta = 2.5 \cdot 10^{-2}$	0.1899 (0.0504)	0.1924 (0.0747)	0.2065 (0.0689)

Notes: *KLIC*-performance values represent the mean (standard deviation in brackets) of $B = 60$ bootstrap replications differing in both the initial parameter values randomly chosen from [–0.3; 0.3] and the data split. *KLIC*(M_1): Learning performance measured in terms of average *KLIC; KLIC*(M_2): Validation performance measured in terms of average *KLIC; KLIC*(M_3): Generalisation performance measured in terms of average *KLIC; M* consists of 992 patterns, M_1 of 496 patterns, M_2 of 248 patterns and M_3 of 248 patterns.

7 Conclusions and Outlook

Learning neural spatial interaction parameters is like solving an unconstrained continuous nonlinear minimisation problem. The task is to find parameter assignments that minimise the given negative log-likelihood function. Product unit neural spatial interaction network learning is a multimodal nonlinear minimisation problem with many local minima. Local minimisation algorithms such as backpropagation of gradient descents have difficulties when the surface of the search space is flat (that is, gradient close to zero), or when the surface is very rugged. When the surface is rugged, a local search from a random starting point generally

converges to a local minimum close to the initial point and to a worse solution than the global minimum.

Global search procedures such as Alopex based search, as opposed to local search, have to be used in learning problems where reaching the global optimum is at premium. But the price one pays for using global search procedures in general and Alopex search in particular is increased computational requirements. The intrinsic slowness of global search procedures is mainly due to the slow but crucial exploration process employed. An important lesson from the results of the study and an interesting avenue for research is, thus, to make global search more speed efficient. This may motivate the development of a hybrid procedure that uses global search to identify regions of the parameter space containing promising local minima and gradient information to actually find them.

References

Bia A. (2000): A study of possible improvements to the Alopex training algorithm. In: *Proceedings of the VIth Brazilian Symposium on Neural Networks*, IEEE Computer Society Press, pp. 125-130

Bishop C.M. (1995): *Neural Networks for Pattern Recognition,* Clarendon Press, Oxford

Cichocki, A. and Unbehauen R. (1993): *Neural Networks for Optimisation and Signal Processing*, John Wiley, Chichester [UK], New York

Efron B. (1982): *The Jackknife, the Bootstrap and Other Resampling Plans*, Society for Industrial and Applied Mathematics SIAM, Philadelphia

Fischer M.M. and Reismann M. (2002a): Evaluating neural spatial interaction modelling by bootstrapping, *Networks and Spatial Economics* 2 (3), 255-268

Fischer M.M. and Reismann M. (2002b): A methodology for neural spatial interaction modelling, *Geographical Analysis* 34 (3), 207-228

Fischer M.M., Reismann M. and Hlavackova-Schindler K. (2003): Neural network modelling of constrained spatial interaction flows: Design, estimation and performance issues, *Journal of Regional Science* 43 (1), 35-61

Harth E. and Pandya A.S. (1988): Dynamics of ALOPEX process: Application to optimization problems. In: Ricciardi L.M. (ed.) *Biomathematics and Related Computational Problems*, Kluwer Academic Publishers, Dordrecht, Boston, London, pp. 459-471

Hassoun M.M. (1995): *Fundamentals of Neural Networks*, MIT Press, Cambridge [MA], London [England]

Kirkpatrick S., Gelatt C.D. Jr. and Vecchi M.P. (1983): Optimization by simulated annealing, *Science* 220, 671-680

Kullback S. and Leibler R.A. (1951): On information and sufficiency, *Annals of Mathematical Statistics* 22 (1), 78-86

Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P. (1992): *Numerical Recipes in C*: *The Art of Scientific Computing*, Cambridge University Press, Cambridge [MA]

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

Rumelhart D.E., Durbin R., Golden R. and Chauvin Y. (1995): Backpropagation: The basic theory. In: Chauvin Y. and Rumelhart D.E. (eds.): *Backpropagation: Theory, Architectures and Applications*, Lawrence Erlbaum Associates, Hillsdale [NJ], pp. 1-34

Unnikrishnan K.P. and Venugopal K.P. (1994): Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks, *Neural Computation* 6 (3), 469-490

14 A Methodology for Neural Spatial Interaction Modelling

with *M. Reismann*

This paper attempts to develop a mathematically rigid and unified framework for neural spatial interaction modelling. Families of classical neural network models, but also less classical ones such as product unit neural network ones are considered for the cases of unconstrained and singly constrained spatial interaction flows. Current practice appears to suffer from least squares and normality assumptions that ignore the true integer nature of the flows and approximate a discrete-valued process by an almost certainly misrepresentative continuous distribution. To overcome this deficiency we suggest a more suitable estimation approach, maximum likelihood estimation under more realistic distributional assumptions of Poisson processes, and utilise a global search procedure, called Alopex, to solve the maximum likelihood estimation problem. To identify the transition from underfitting to overfitting we split the data into training, internal validation and test sets. The bootstrapping pairs approach with replacement is adopted to combine the purity of data splitting with the power of a resampling procedure to overcome the generally neglected issue of fixed data splitting and the problem of scarce data. In addition, the approach has power to provide a better statistical picture of the prediction variability. Finally, a benchmark comparison against the classical gravity models illustrates the superiority of both, the unconstrained and the origin-constrained neural network model versions in terms of generalisation performance measured by Kullback and Leibler's information criterion.

1 Introduction

There are several phases that an emerging field goes through before it reaches maturity, and GeoComputation is no exception. There is usually a trigger for birth of the field. In our case, new techniques such as neural networks and evolutionary computation, significant progress in computing technology, and the emerging data-rich environment inspired many scholars to revisit old and tackle new spatial problems. The result has been a wealth of new approaches, with significant improvements in many cases (see Longley et al. 1998, Fischer and Leung 2001).

After the initial excitement settles in, the crest-breaking question is whether the new community of researchers can produce sufficient results to sustain the field, and whether practitioners will find these results to be of quality, novelty, and relevance to make a real impact. Successful applications of geocomputational models and techniques to a variety of problems such as data mining, pattern recognition,

optimisation, traffic forecasting, and spatial interaction modelling rang the bell signifying the entry of GeoComputation as an established field.

This paper is a response to the perceived omission in the comprehensive understanding of one of the most important subfields in GeoComputation. While various papers on neural network modelling of unconstrained spatial interaction flows have appeared in the past decade, there has yet to be an advanced discussion of the general concepts involved in the application of such models. This paper attempts to fill the gap. Among the elements which should be of interest to those interested in applications are estimation and performance issues.

The paper proceeds as follows. The first section points to some shortcomings evident in current practice and motivates to depart in two directions: First, to employ maximum likelihood under more realistic distributional assumptions rather than least squares and normality assumptions, and second to utilise bootstrapping to overcome the problems of fixed data splitting and the scarcity of data that affect performance and reliability of the model results. Section 2 describes classical unconstrained neural spatial interaction models and less classical ones. Classical models are those that are constructed using a single hidden layer of summation units. In these network models each input to the hidden node is multiplied by a weight and then summed. Less classical models utilise a product unit rather than the standard summation neural network framework for modelling interactions over space.

Unconstrained – summation unit and product unit – neural spatial interaction models represent rich and flexible families of spatial interaction function approximators. But they may be of little practical value if a priori information is available on accounting constraints on the predicted flows. Section 3 moves to the case of constrained spatial interaction. To satisfactorily tackle this issue within a neural network environment it is necessary to embed the constraint-handling mechanism within the model structure. This is a far from easy task. We briefly describe the only existing generic model approach for the single constrained case (see Fischer et al. 2001), and present summation and product unit model versions. We reserve the doubly constrained case to subsequent work.

We view parameter estimation (network learning) in an optimisation context and develop a rationale for an appropriate objective (loss) function for the estimation approach in Section 4. Global search procedures such as simulated annealing or Alopex may be employed to solve the maximum likelihood estimation problem. We follow Fischer et al. (2001) to utilise the Alopex procedure that differs from the method of simulated annealing in three important aspects. First, correlations between changes in individual parameters and changes in the loss function are used rather than changes in the loss function only. Second, all parameter changes are accepted at every iteration, and, third, during an iteration step all parameters are updated simultaneously.

The standard approach to evaluate the generalisation performance of neural network models is to split the data set into three subsets: the training set, the internal validation set, and the testing set. It has become common practice to fix these sets. A bootstrapping approach is suggested to overcome the generally neglected problem of sensitivity to the specific splitting of the data, and to get a better statistical

picture of prediction variability of the models. Section 5 illustrates the application of the various families of neural spatial interaction function approximators discussed in the previous sections and presents the results of a comparison of the performance of the summation and the product unit neural network model versions (unconstrained and origin-constrained cases) against the corresponding standard gravity models. The testbed for the evaluation uses interregional telecommunication traffic data from Austria. Section 6 outlines some directions for future research.

2 Departures from Current Practice

We will begin our analysis with the simplest case, namely that of unconstrained spatial interaction. For concreteness and simplicity, we consider neural spatial interaction models based on the theory of single hidden layer feedforward models. Current research in this field appears to suffer from least squares and Gaussian assumptions that ignore the true integer nature of the flows and approximate a discrete-valued process by an almost certainly misrepresentative distribution. As a result, least squares estimates and their standard errors can be seriously distorted. To overcome this shortcoming we will develop a more appropriate estimation approach under more realistic distributional assumptions.

Thus, throughout the paper we assume observations generated as the realisation of a sequence $\{Z_k = (X_k, Y_k), k = 1, ..., K\}$ of $(N+1)\times 1$ vectors $(N \in \mathbb{N})$ defined on a Poisson probability space. The random variables Y_k represent targets. Their relationship to the variables X_k is of primary interest. When $E(Y_k) < \infty$, the conditional expectation of Y_k given X_k exists, denoted as $g = E(Y_k \mid X_k)$. Defining $\varepsilon_k \equiv Y_k - g(X_k)$, we can also write $Y_k = g(X_k) + \varepsilon_k$. The unknown spatial interaction function g embodies the systematic part of the stochastic relation between Y_k and X_k. The error ε_k is noise, with the property that $E(\varepsilon_k \mid X_k) = 0$ by construction. Our problem is to learn (estimate, approximate) the mapping g from a realisation of the sequence $\{Z_k\}$.

In practice, we observe a realisation of only a finite part of the sequence $\{Z_k\}$, a training set or sample of size K (that is, a realisation of $\{Z_k \mid k = 1, ..., K\}$). Because g is an element of a space of spatial interaction functions, say $\mathcal{G}$, we have essentially no hope of learning g in any complete sense from a sample of fixed finite size. Nevertheless, it is possible to approximate g to some degree of accuracy using a sample of size K, and to construct increasingly accurate approximations with increasing K. We will refer to such a procedure interchangeably as learning, estimation, or approximation.

There are many standard procedures of function approximation to this function g. Perhaps the simplest is linear regression. Since feedforward neural networks are characteristically nonlinear it is useful to view them as performing a kind of nonlinear regression. Several of the issues that come up in regression analysis are also relevant to the kind of nonlinear regression performed by neural networks.

One important example comes up in the cases of *underfitting* and *overfitting*. If the neural network model is able to approximate only a narrow range of functions, then it may be incapable of approximating the true spatial interaction function no matter how much training data is available. Thus, the model will be biased, and it is said to be *underfitted*. The solution to this problem seems to be to increase the complexity of the neural network, and, thus, the range of spatial interaction functions, that can be approximated, until the bias becomes negligible. But, if the complexity (too many degrees of freedom) rises too far, then *overfitting* may arise and the fitted model will again lead to poor estimates of the spatial interaction function. If overfitting occurs, then the fitted model can change significantly as single training samples are perturbed and, thus shows high variance. The ultimate measure of success is not how closely the model approximates the training data, but how well it accounts for not yet seen cases. Optimising the generalisation performance requires that the neural network complexity is adjusted to minimise both the bias and the variance as much as possible.

Since the training data will be fitted more closely as the model complexity increases, the ability of the trained model to predict this data cannot be utilised to identify the transition from underfitting to overfitting. In order to choose a suitably complex model, some means of directly estimating the generalisation performance are needed. For neural spatial interaction models data splitting is commonly used. Though this procedure is simple to use in practice, effective use of data splitting may require a significant reduction in the amount of data which is available to train the model. If the available data is limited and sparsely distributed – and this tends to be the rule rather than the exception in spatial interaction contexts, then any reduction in amount of training data may obscure or remove features of the true spatial interaction function from the training set.

In this contribution, we address this issue by adopting the bootstrapping pairs approach (see Efron 1982) with replacement. This approach will combine the purity of data splitting with the power of a resampling procedure and, moreover, will allow a better statistical picture of the prediction variability. An additional benefit of the bootstrap is that it provides approximations to the sampling distribution of the test statistic of interest that are considerably more accurate than the analytically obtained large sample approximations. Formal investigation of this additional benefit is beyond the scope of this contribution. We have a full agenda just to analyse the performance of summation and product unit neural network models for the cases of unconstrained and constrained spatial interaction. But we anticipate that our bootstrapping procedure may well afford such superior finite sample approximations.

3 Families of Unconstrained Neural Spatial Interaction Models

In many spatial interaction contexts, little is known about the form of the spatial interaction function which is to be approximated. In such cases it is generally not

possible to use a parametric modelling approach where a mathematical model is specified with unknown coefficients which have to be estimated to fit the model. The ability of neural spatial interaction models to model a wide range of spatial interaction functions relieves the model user of the need to specify exactly a model that includes all the necessary terms to model the true spatial interaction function.

The Case of Unconstrained Spatial Interaction

There is a growing literature in geography and regional science that deals with alternative model specifications and estimators for solving unconstrained spatial interaction problems. Examples include, among others, Fischer and Gopal (1994), Black (1995), Nijkamp et al. (1996), Bergkvist and Westin (1997), Bergkvist (2000), Reggiani and Tritapepe (2000), Thill and Mozolin (2000), Mozolin et al. (2000). All these models are members of the following *general class of unconstrained neural spatial interaction models* given by

$$\Omega^H(\boldsymbol{x}, \boldsymbol{w}^H) = \psi\left(w_{00} + \sum_{h=1}^{H} w_{0h}\, \varphi\left(\sum_{n=1}^{N} w_{1hn}\, x_n \right) \right) \tag{1}$$

where the *N*-dimensional euclidean space (generally, $N = 3$) is the input space and the one-dimensional euclidean space the output space. Vector $\boldsymbol{x} = (x_1, ..., x_N)$ is the input vector that represents measures characterising the origin and the destination of spatial interaction as well as their separation. $\boldsymbol{w}^H \equiv (\boldsymbol{w}_0, \boldsymbol{w}_1)$ is the $(HN + H + 1) \times 1$ vector of the network weights (parameters). There are H hidden units. The vector $\boldsymbol{w}_0$ contains the hidden to output unit weights, $\boldsymbol{w}_0 \equiv (w_{00}, w_{01}, ..., w_{0H})$, and the vector $\boldsymbol{w}_1$ contains the input to hidden unit weights, $\boldsymbol{w}_1 \equiv (w_{10}, ..., w_{1H})$ with $w_{1h} \equiv (w_{1h1}, ..., w_{1hN})$. We allow a bias at the hidden layer by including w_{00}. A bias at the input array may be taken into consideration by setting $x_1 \equiv 1$, then $N = 4$. φ is a hidden layer transfer function, ψ an output unit transfer function, both continuously differentiable of order 2 on $\mathfrak{R}$. Note that the model output function and the weight vector are explicitly indexed by the number, *H*, of hidden units in order to indicate the dependence. But to simplify notation we drop the superindex hereafter.

Figure 1 depicts the corresponding network architecture. Hidden units, denoted by the symbol Σ, indicate that each input is multiplied by a weight and then summed. Thus, models of type (1) may be termed unconstrained *summation unit* neural spatial interaction models. The family of approximations (1) embodies several concepts already familiar from the pattern recognition literature. It is the combination of these that is novel. Specifically, $\Sigma_{n=1}^{N} w_{1hn}\, x_n$ is a familiar linear discriminant function (see Young and Calvert 1974) which – when transformed by φ – acts as a nonlinear feature detector. The 'hidden' features are then subjected to a linear discriminant function and filtered through ψ. The approximation

benefits from the use of nonlinear feature detectors, while retaining many of the advantages of linearity in a particularly elegant manner.

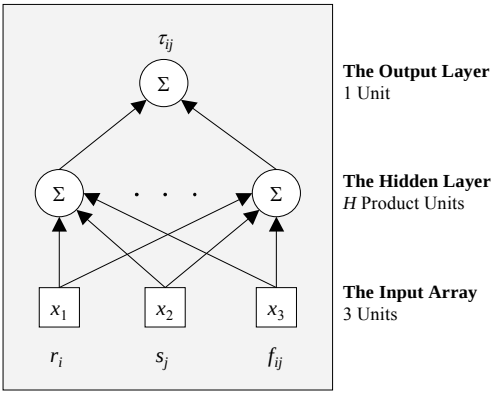


Figure 1 Architecture of the unconstrained summation unit neural spatial interaction models as defined by Equation (1) for $N = 3$

A leading case occurs when both transfer functions are specified as logistic functions.[1] This leads to the model

$$\Omega_L(\boldsymbol{x},\boldsymbol{w}) = \left\{1 + \exp\left[-\left(w_{00} + \sum_{h=1}^{H} w_{0h}\left(1 + \exp\left(-\sum_{n=1}^{N} w_{1hn}\, x_n\right)\right)^{-1}\right)\right]\right\}^{-1} \tag{2}$$

that has been often used in practice (see, for example, Mozolin et al. 2000, Fischer et al. 1999, Fischer and Leung 1998, Gopal and Fischer 1993, 1996, Black 1995, Fischer and Gopal 1994, Openshaw 1993).

Product Unit Model Versions

Neural spatial interaction models of type (1) are constructed using a single hidden layer of summation units. In these networks each input to the hidden node is multiplied by a weight and then summed. A nonlinear transfer function, such as the

[1] Sigmoid transfer functions such as the logistic function are somewhat better behaved than many other functions with respect to the smoothness of the error surface. They are well behaved outside of their local region in that they saturate and are constant at zero or one outside the training region. Sigmoidal units are roughly linear for small weights (net input near zero) and get increasingly nonlinear in their response as they approach their points of maximum curvature on either side of the midpoint.

logistic function, is employed at the hidden layer. Neural network approximation theory has shown the attractivity of such summation networks.

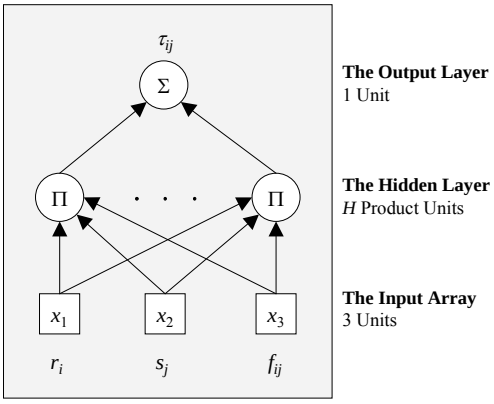


Figure 2 Architecture of the unconstrained product unit neural spatial interaction models as defined by Equation (3) for $N = 3$

In the neural network community it is well known that supplementing the inputs to a neural network model with higher-order combinations of the inputs increases the capacity of the network in an information capacity sense (see Cover 1965) and its ability to learn (see Giles and Maxwell 1987). This may motivate to use of a product unit rather than the standard summation unit neural network framework for modelling interactions over space. The *general class of unconstrained product unit neural spatial interaction models* is given as

$$^{\pi}\Omega(\boldsymbol{x},\boldsymbol{w}) = \psi\left(w_{00} + \sum_{h=1}^{H} w_{0h} \ \varphi\left(\prod_{n=1}^{N} x_n^{w_{1hn}} \right) \right) \tag{3}$$

which contains both product and summation units. The product units compute the product of inputs, each raised to a variable power. Figure 2 illustrates the corresponding network architecture.

Specifying $\varphi(\cdot)$ to be the identity function and $\psi(\cdot)$ to be the logistic function we obtain the following special case of (3)

$$^{\pi}\Omega_L(\boldsymbol{x},\boldsymbol{w}) = \left\{ 1 + \exp\left[- \left(w_{00} + \sum_{h=1}^{H} w_{0h} \left(\prod_{n=1}^{N} x_n^{w_{1hn}} \right) \right) \right] \right\}^{-1} \tag{4}$$

4 Neural Networks of Constrained Spatial Interaction Flows

Classical neural network models of the form (1) and less classical models of the type (3) represent rich and flexible families of neural spatial interaction approximators. But they may be of little practical value if a priori information is available on accounting constraints of the predicted flows. For this purpose Fischer et al. (2001) have recently developed a novel class of neural spatial interaction models that are able to deal efficiently with the singly constrained case of spatial interaction.

The models are based on a modular connectionist architecture that may be viewed as a linked collection of functionally independent modules with identical feedforward topologies (two inputs, *H* hidden product units, and a single summation unit), operating under supervised learning algorithms. The prediction is achieved by combining the outcome of the individual modules using a nonlinear output transfer function multiplied with a bias term that implements the accounting constraint.

Without loss of generality we consider the origin-constrained model version only. Figure 3 illustrates the modular network architecture of the models. Modularity is seen here as decomposition on the computational level. The network is composed of two processing layers and two layers of network parameters. The first processing layer is involved with the extraction of features from the input data. This layer is implemented as a layer of *J* functionally independent modules with identical topologies. Each module is a feedforward network with two inputs x_{2j-1} and x_{2j} (representing measures of destination attractiveness and separation between origin and destination, respectively), *H* hidden product units, denoted by the symbol Π, and terminates with a single summation unit, denoted by the symbol Σ. The collective output of these modules constitutes the input to the second processing layer consisting of *J* output units that perform the flow prediction.

This network architecture implements the general *class of product unit neural models of origin-constrained* [OC] *spatial interaction*

$$ {}^{\pi}\Omega^{OC}(\boldsymbol{x},\boldsymbol{w})_j \;=\; \psi_j\left(\sum_{h=1}^{H}\gamma_h\,\varphi_h\left(\prod_{n=2j-1}^{2j} x_n^{\beta_{hn}}\right)\right) \qquad j=1,\ldots,J \tag{5} $$

with $\varphi_h: \Re \to \Re$, $\psi_j: \Re \to \Re$ and $\boldsymbol{x} \in \Re^{2J}$, that is, $\boldsymbol{x} = (x_1, x_2, ..., x_{2j-1}, x_{2j}, ..., x_{2J-1}, x_{2J})$ where x_{2j-1} represents a variable pertaining to destination j (j = 1, ..., J) and x_{2j} a variable f_{ij} pertaining to the separation from region i to region j(i = 1, …, I; j = 1, …, J) of the spatial interaction system under scrutiny. β_{hn} (h = 1; ..., H; n = 2j – 1, 2j) are the input-to-hidden connection weights, and γ_h (h = 1; ..., H) the hidden-to-output weights in the jth module of the network model. The symbol $\boldsymbol{w}$ is a convenient shorthand notation of the (3H)-dimensional vector of all the model parameters. ψ_j ($j = 1, ..., J$) represents a nonlinear summation unit transfer function and φ_h ($h = 1, ..., H$) a linear hidden product unit transfer function.

Specifying $\varphi_h(\cdot)$ to be the identity function and $\psi_j(\cdot)$ a nonlinear normalised function we obtain the following important special case of (5)

$$ {}^{\pi}\Omega_1^{OC}(\boldsymbol{x},\boldsymbol{w})_j \;=\; \tilde{b}_{(i)} \frac{\sum_{h=1}^{H} \gamma_h \prod_{n=2\,j-1}^{2\,j} x_n^{\beta_{hn}}}{\sum_{j'=1}^{J}\sum_{h'=1}^{H} \gamma_{h'} \prod_{n=2\,j'-1}^{2\,j'} x_n^{\beta_{h'n}}} \qquad j = 1, ..., J \tag{6} $$

where $\tilde{b}_{(i)}$ is the bias signal that can be thought as being generated by a 'dummy unit' whose output is clamped at the scalar $t_{i\bullet}$. A more detailed description of the model may be found in Fischer et al. (2001).

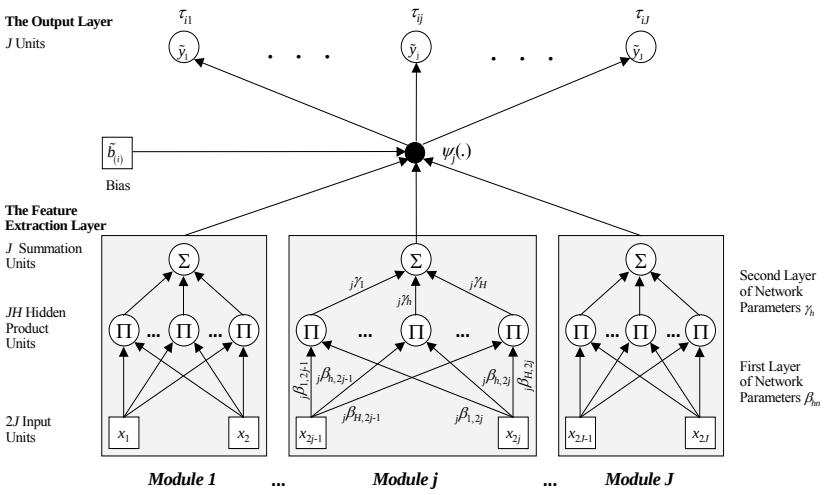


Figure 3 Origin-constrained product unit neural spatial interaction models as defined by Equation (5)

Summation Unit Model Versions

The summation unit version of the *general class of product unit neural network models of origin-constrained spatial interaction* may be easily derived from Equation (5):

$$ {}^{\Sigma}\Omega^{OC}(\boldsymbol{x},\boldsymbol{w})_j \;=\; \psi_j\left(\sum_{h=1}^{H} \gamma_h\, \varphi_h\left(\sum_{n=2\,j-1}^{2\,j} \beta_{hn}\, x_n\right)\right) \qquad j = 1, ..., J \tag{7} $$

where $\varphi_h: \Re \to \Re$, $\psi_j: \Re \to \Re$ and $\boldsymbol{x} \in \Re^{2J}$ as above. Specifying $\varphi_h(\cdot)$ as logistic function for $h = 1,...,N$, and $\psi_j(\cdot)$ as nonlinear normalised output transfer function we obtain the following *origin-constrained member* of class (7):

$$ {}^{\Sigma}\Omega_1^{OC}(\boldsymbol{x},\boldsymbol{w})_j = \tilde{b}_{(i)} \frac{\sum_{h=1}^{H} \gamma_h \left(1 + \exp\left(-\sum_{n=2j-1}^{2j} \beta_{hn}\, x_n\right)\right)^{-1}}{\sum_{j'=1}^{J}\sum_{h'=1}^{H} \gamma_{h'} \left(1 + \exp\left(-\sum_{n=2j'-1}^{2j'} \beta_{h'n}\, x_n\right)\right)^{-1}} \qquad j = 1, ..., J \quad (8) $$

5 A Rationale for the Estimation Approach

If we view a neural spatial interaction model, unconstrained or constrained, as generating a family of approximations (as $\boldsymbol{w}$ ranges over $\boldsymbol{W}$, say) to a spatial interaction function g, then we need a way to pick a best approximation from this family. This is the function of network learning (training, parameter estimation) which might be viewed as an optimisation problem.

We develop a rationale for an appropriate objective (loss, cost) function for this task. Following Rumelhart et al. (1995) we propose that the goal is to find that model which is the most likely explanation of the observed data set, say *M*. We can express this as attempting to maximise the term

$$ P(\Omega(\boldsymbol{w}) \mid M) = \frac{P(M \mid \Omega(\boldsymbol{w}))\; P(\Omega(\boldsymbol{w}))}{P(M)} \quad (9) $$

where Ω represents the neural spatial interaction model (with all the weights $\boldsymbol{w}^H$) in question, unconstrained or constrained. $P(M \mid \Omega(\boldsymbol{w}))$ is the probability that the model would have produced the observed data M. Since sums are easier to work with than products, we will maximise the log of this probability, and since this log is a monotonic transformation, maximising the log is equivalent to maximising the probability itself. In this case we have

$$ \ln P(\Omega(\boldsymbol{w}) \mid M) = \ln P(M \mid \Omega(\boldsymbol{w})) + \ln P(\Omega(\boldsymbol{w})) - \ln P(M). \quad (10) $$

The probability of the data, $P(M)$, is not dependent on the model. Thus, it is sufficient to maximise $\ln P(M \mid \Omega(\boldsymbol{w})) + \ln P(\Omega(\boldsymbol{w}))$. The first of these terms represents the probability of the data given the model, and hence measures how well the network accounts for the data. The second term is a representation of the model itself; that is, it is a prior probability, that can be utilised to get information and constraints into the learning procedure.

We focus solely on the first term, the performance, and begin by noting that the data can be broken down into a set of observations, $M = \{z_k = (x_k, y_k) \mid k = 1, ..., K\}$, each z_k, we will assume, chosen independently of the others. Hence we can write the probability of the data given the model as

$$\ln P(M \mid \Omega(\boldsymbol{w})) = \ln \prod_k P(z_k \mid \Omega(\boldsymbol{w})) = \sum_k \ln P(z_k \mid \Omega(\boldsymbol{w})). \tag{11}$$

Note that this assumption permits to express the probability of the data given the model as the sum of terms, each term representing the probability of a single observation given the model. We can still take another step and break the data into two parts: the observed input data x_k and the observed target y_k. Therefore we can write

$$\ln P(M \mid \Omega(\boldsymbol{w})) = \sum_k \ln P(y_k \mid x_k \text{ and } \Omega(\boldsymbol{w})_k) + \sum_k \ln P(x_k). \tag{12}$$

Since we assume that x_k does not depend on the model, the second term of the equation will not affect the determination of the optimal model. Thus, we need only to maximise the term $\Sigma_k \ln P(y_k \mid x_k \text{ and } \Omega(\boldsymbol{w})_k)$.

Up to now we have – in effect – made only the assumption of the independence of the observed data. In order to proceed, we need to make some more specific assumptions, especially about the relationship between the observed input data x_k and the observed target data y_k, a probabilistic assumption. We assume that the relationship between x_k and y_k is not deterministic, but that for any given x_k there is a distribution of possible values of y_k. But the model is deterministic, so rather than attempting to predict the actual outcome we only attempt to predict the expected valued of y_k given x_k. Therefore, the model output is to be interpreted as the mean bilateral interaction frequencies (that is, those from the region of origin to the region of destination). This is, of course, the standard assumption.

To proceed further, we have to specify the form of the distribution of which the model output is the mean. Of particular interest to us is the assumption that the observed data are the realisation of a sequence of independent Poisson random variables. Under this assumption we can write the probability of the data given the model as

$$P(y_k \mid x_k \text{ and } \Omega(\boldsymbol{w})_k) = \frac{\prod_k \Omega(\boldsymbol{w})_k^{y_k} \exp(-\Omega(\boldsymbol{w})_k)}{y_k!} \tag{13}$$

and, hence, define a maximum likelihood estimator as a parameter vector $\hat{\boldsymbol{w}}$ which maximises the log-likelihood L

$$\max_{\boldsymbol{w} \in \boldsymbol{W}} L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) = \max_{\boldsymbol{w} \in \boldsymbol{W}} \sum_k (y_k \ln \Omega(\boldsymbol{w})_k - \Omega(\boldsymbol{w})_k). \tag{14}$$

Instead of maximising the log-likelihood it is more convenient to view learning as solving the minimisation problem

$$\min_{\boldsymbol{w} \in \boldsymbol{W}} \lambda(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) = \min_{\boldsymbol{w} \in \boldsymbol{W}} \left[- L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) \right] \tag{15}$$

where the loss (cost) function λ is the negative log-likelihood L. λ is non-negative, continuously differentiable on the Q-dimensional parameter space ($Q = HN + H + 1$ in the unconstrained case and $Q = 3H$ in the constrained one) which is a finite dimensional closed bounded domain and, thus, compact. It can be shown that λ assumes its value as the weight minimum under certain conditions.

6 Training the Neural Network Models

Since the loss function λ is a complex nonlinear function of $\boldsymbol{w}$ for the neural spatial interaction models, $\hat{\boldsymbol{w}}$ cannot be found analytically and computationally intensive iterative optimisation techniques such as global search procedures must be utilised to find (15). Simulated annealing, genetic algorithms and the Alopex[2] procedure are attractive candidates for this task. We utilise the latter as described in Fischer et al. (2001).

The loss function $\lambda(\boldsymbol{w})$ is minimised by means of weight changes that are computed for the sth step $(s > 2)$ of the iteration process as follows,[3]

$$w_k(s) = w_k(s-1) + \delta_k(s) \tag{16}$$

where $\delta_k(s)$ is a small positive or negative step of size δ with the following properties:

$$\delta_k(s) = \begin{cases} -\delta \text{ with probability } p_k(s) \\ +\delta \text{ with probability } \left[1 - p_k(s)\right]. \end{cases} \tag{17}$$

The probability $p_k(s)$ for a negative step is given by the Boltzmann distribution

$$p_k(s) = \left[1 + \exp\left\{ - \frac{C_k(s)}{T(s)} \right\} \right]^{-1} \tag{18}$$

where

[2] Alopex is an acronym for **al**gorithm f**o**r **p**attern **ex**traction.

[3] For the first two iterations, the weights are chosen randomly.

$$C_k(s) = \Delta w_k(s)\ \Delta\lambda(s) \tag{19}$$

with

$$\Delta w_k(s) = w_k(s-1) - w_k(s-2) \tag{20}$$

and

$$\Delta\lambda(s) = \lambda(s-1) - \lambda(s-2). \tag{21}$$

The parameter T in Equation (18), termed temperature in analogy to simulated annealing, is updated using the following annealing schedule:

$$T(s) = \begin{cases} \frac{\delta}{QS} \sum_k \sum_{s'=s-S}^{s-1} |C_k(s')| & \text{if } s \text{ is a multiple of } S \\ T(s-1) & \text{otherwise} \end{cases} \tag{22}$$

where ($Q = HN + H + 1$ in the case of the unconstrained models, and $Q = 3H$ in the case of the constrained models) denotes the number of weights. When T is small, the probability of changing the parameters is around zero if C_k is negative and around one if C_k is positive. If T is large, then $p_k \cong 0.5$ (see Bia 2000).

The effectiveness of Alopex in locating global minima and its speed of convergence critically depend on the balance of the size of the feedback term $\Delta w_k\ \Delta\lambda$, and the temperature T. If T is very large compared to $\Delta w_k\ \Delta\lambda$ the process does not converge. If T is too small, a premature convergence to a local minimum might occur. The procedure is governed by three parameters: the initial temperature T, the number of iterations, S, over which the correlations are averaged for annealing, and the step size δ. The temperature T and the S-iterations cycles seem to be of secondary importance for the final performance of the algorithm. The initial temperature T may be set to a large value of about 1,000. This allows the algorithm to get an estimate of the average correlation in the first S iterations and reset it to an appropriate value according to Equation (22). S may be chosen between 10 and 100. In contrast to T and S, δ is a critical parameter that has to be selected heuristically with care. There is no way to a priori identify δ in the case of multimodal parameter spaces.

The Termination Criterion

It has been observed that forceful training may not produce network models with adequate generalisation ability, although the learning error achieved is small. The most common remedy for this problem is to monitor model performance during

training to assure that further training improves generalisation as well. For this purpose an additional set of validation data, independent from the training data is used. In a typical training phase, it is normal for the validation error to decrease. This trend may not be permanent, however. At some point the validation error usually reverses. Then the training process should be stopped. In our implementation of the Alopex procedure network training is stopped when $\kappa = 40,000$ consecutive iterations are unsuccessful. κ has been chosen so large at the expense of the greater training time, to ensure more reliable estimates.

7 Experimental Environment, Performance Tests and Benchmark Comparisons

To illustrate the application of modelling and estimation tools discussed in the previous sections we utilise interregional telecommunication traffic data from Austria and standard gravity models as benchmarks.

The Benchmark Models

The *standard unconstrained gravity model*

$$\tau_{ij}^{grav} = k \; r_i^{\alpha} \; s_j^{\beta} \; d_{ij}^{-\gamma} \qquad i = 1, ..., I; \; j = 1, ..., J; \; j \neq i \tag{23}$$

with

$$k = \frac{t_{\bullet\bullet}}{\sum_{i \neq j} \sum_{j} r_i^{\alpha} \; s_j^{\beta} \; d_{ij}^{-\gamma}} \tag{24}$$

$$t_{\bullet\bullet} := \sum_{i \neq j} \sum_{j} t_{ij} \tag{25}$$

serves as a benchmark model for the unconstrained neural spatial interaction models[4], that is, the classical models of type (2) and the less classical ones of type (4). τ_{ij}^{grav} denotes the estimated flow from i to j, k is a factor independent of all origins and destinations, α reflects the relationship of r_i with τ_{ij}^{grav} and β the relationship of s_j with τ_{ij}^{grav}. γ is the distance sensitivity parameter, $\gamma > 0$. r_i and s_j are measured in terms of the gross regional product, d_{ij} in terms of distances

[4] There is virtual unanimity of opinion that site-specific variables, such as s_j in this case, are generally best represented as power functions. The specification of f_{ij} is consistent with general consensus that the power function is more appropriate for analysing longer distance interactions (Fotheringham and O'Kelly 1989).

from *i* to *j*, whereas t_{ij} in terms of erlang (see Fischer and Gopal 1994 for more details).

The *standard origin-constrained gravity model*

$$^{orig}\tau_{ij}^{grav} = b_{(i)}\ s_j^{\alpha}\ d_{ij}^{-\gamma} \qquad i = 1, ..., I;\ j = 1, ..., J;\ j \neq i \tag{26}$$

with

$$b_{(i)} = \frac{t_{i\bullet}}{\sum_{j \neq i} s_j^{\alpha}\ d_{ij}^{-\gamma}} \tag{27}$$

where

$$t_{i\bullet} := \sum_{j \neq i} t_{ij} \tag{28}$$

is used as benchmark model for the constrained neural spatial interaction models (6) and (8). $b_{(i)}$ is the origin-specific balancing factor. $\alpha, \gamma, s_j, d_{ij}$ and t_{ij} are defined as above.

Performance Measure

One needs to be very careful when selecting a measure to compare different models. It makes not much sense to utilise least squares related performance measures, such as the average relative variances or the standardised root mean square, in the context of our ML estimation approach. Model performance is measured in this study by means of Kullback and Leibler's (1951) information criterion (*KLIC*) which is a natural performance criterion for the goodness-of-fit of ML estimated models:

$$KLIC(M) = \sum_{u=1}^{U} \frac{y_u}{\sum_{u'=1}^{U} y_{u'}} \ln\left[\frac{y_u \left(\sum_{u'=1}^{U} y_{u'} \right)^{-1}}{\Omega(x_u, \boldsymbol{w}) \left[\sum_{u'=1}^{U} \Omega(x_{u'}, \boldsymbol{w}) \right]^{-1}} \right] \tag{29}$$

where (x_u, y_u) denotes the *u*th pattern of the data set M, and Ω is the neural spatial interaction model under consideration. The performance measure has a minimum at zero and a maximum at positive infinity when $y_u > 0$ and $\Omega(x_u, \boldsymbol{w}) = 0$ for any (x_u, y_u) pair.

The Data, Data Splitting and Bootstrapping

To model interregional telecommunication flows for Austria we utilise three Austrian data sources – a (32, 32)-interregional telecommunication flow matrix (t_{ij}), a (32, 32)-distance matrix (d_{ij}), and gross regional products for the thirty-two telecommunication regions – to produce a set of 992 4-tupel $(r_i, s_j, d_{ij}; t_{ij})$ with $i, j = 1, ..., 32 \quad (i \neq j)$. The first three components represent the input vector of the unconstrained models, the second and the third components represent the input variables x_{2j-1} and x_{2j} of the *j*th module of the origin-constrained network models, and the last component the target output. The bias term $\tilde{b}_{(i)}$ is clamped to the scalar $t_{i\bullet}$. s_j represents the potential draw of telecommunication in *j* and is measured in terms of the gross regional product, d_{ij} denotes distances from *i* to *j*, while t_{ij} represents telecommunication traffic flows. The input data[5] were rescaled to lie in [0.1, 0.9].

The telecommunication data stem from network measurements of carried traffic in Austria in 1991, in terms of erlang, an internationally widely used measure of telecommunication contact intensity, which is defined as the number of phone calls (including facsimile transfers) multiplied by the average length of the call (transfer) divided by the duration of measurement (for more details, see Fischer and Gopal 1994). The data refer to the telecommunication traffic between the thirty-two telecommunication districts representing the second level of the hierarchical structure of the Austrian telecommunication network. Due to measurement problems, intraregional traffic (i.e. $i = j$) is left out of consideration.

The standard approach to evaluate the out-of-sample (prediction) performance of a neural spatial interaction model (see Fischer and Gopal 1994) is to split the total data set *M* of 992 samples into three subsets: the *training* (in-sample) *set* $M_1 = \{(x_{u1}, y_{u1})\ \{\text{with } u1 = 1, ..., U_1 = 496 \text{ patterns}\}$, the *internal validation set* $M_2 = \{(x_{u2}, y_{u2})\ \{\text{with } u2 = 1, ..., U_2 = 248 \text{ patterns}\}$ and the *testing* (prediction, out-of-sample) *set* $M_3 = \{(x_{u3}, y_{u3})\ \text{with } u3 = 1, ..., U_3 = 248 \text{ patterns}\}$. M_1 is used only for parameter estimation, while M_2 for validation. The generalisation performance of the model is assessed on the testing set M_3. It has become common practice to fix these sets. But recent experience has found this approach to be very sensitive to the specific splitting of the data. To overcome this problem as well as the problem of scarce data we make use of the bootstrapping pairs approach (Efron 1982) with replacement. This approach combines the purity of splitting the data into three disjoint data sets with the power of a resampling procedure and allows us also to get a better statistical picture of the prediction variability.

The idea behind this approach is to generate *B* pseudoreplicates of the training, validation and test sets, then to reestimate the model parameters $\boldsymbol{w}$ on each training bootstrap sample, stopping training on the basis of the validation and testing out-of-sample performance of the test bootstrap samples. In this bootstrap world,

[5] In the case of the summation unit model versions the input data were preprocessed to logarithmically transformed data scaled into [0.1, 0.9].

the errors of prediction and the errors in the parameter estimates are directly observable. Statistics on parameter reliability can easily be computed.

Implementing the approach involves the following steps (see Fischer and Reismann 2002):

Step 1: Conduct three totally independent resampling operations in which *B* independent *training bootstrap samples*, *B* independent *validation bootstrap samples* and *B* independent *testing bootstrap sets* are genearted, by randomly sampling U_1, U_2 and U_3 times, respectively, with replacement from the observed input-output pairs *M*.

Step 2: For each training bootstrap sample the minimisation problem (15) is solved by applying the Alopex procedure. During the training process the *KLIC*-performance of the model is monitored on the corresponding bootstrap validation set. The training process is stopped as specified in Section 5.

Step 3: Calculate the *KLIC*-statistic of generalisation performance for each test bootstrap sample. The distribution of the pseudoerrors can be computed, and used to approximate the distribution of the real errors. This approximation is the bootstrap.

Step 4: The variability of the *B* bootstrap *KLIC*-statistics gives an estimate of the expected accuracy of the model performance. Thus, the standard errors of the generalisation performance statistic is given by the sample standard deviation of the *B* bootstrap replications.

Performance Tests and Results

We consider first

- the summation unit neural network Ω_L [see Equation (2)] and
- the product unit neural network ${}^{\pi}\Omega_L$ [see Equation (4)], to model the unconstrained case of spatial interaction, and then
- the modular product unit neural network version ${}^{\pi}\Omega_1^{OC}$ [see Equation (6)] and
- the modular summation unit neural network version ${}^{\Sigma}\Omega_1^{OC}$ [see Equation (8)]

of singly constrained neural spatial interaction models to model the origin-constrained case. Conventional gravity model specifications [see Equations (23)-(25) for the unconstrained case and Equations (26)-(28) for the origin-constrained case] serve as benchmark models.

All the models were calibrated by means of the ML-estimation approach utilising the Alopex procedure to eliminate the effect of different estimation procedures on the result. In order to do justice to each model specification, the critical

Alopex parameter δ (step size) was systematically sought for each model. The Alopex parameters T and S were set to 1,000 and 10, respectively. We made use of the bootstrapping pairs approach (B = 60) to overcome the problem of sensitivity to the specific splitting of the data into in-sample, internal validation and generalisation data sets, and the scarcity of data, but also to get a better statistical picture of prediction variability.

Table 1 The unconstrained case of spatial interaction – summation unit and product unit neural networks, Ω_L and ${}^{\pi}\Omega_L$ [see Equations (2) and (4)], estimated by the Alopex procedure: The choice of H and δ [T = 1,000; S = 10]

		Summation unit network			*Product unit network*		
		$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$	$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$
H = 2	δ = 0.0005	0.2396 (0.0619)	0.2555 (0.1934)	0.2744 (0.1718)	**0.2931** (0.0763)	**0.2784** (0.1398)	0.3228 (0.1623)
	δ = 0.0010	0.2415 (0.0565)	**0.2334** (0.1426)	**0.2535** (0.1334)	0.2955 (0.0740)	0.2847 (0.1335)	0.3199 (0.1635)
	δ = 0.0050	**0.2327** (0.0545)	0.2418 (0.1427)	0.2605 (0.1356)	0.3084 (0.0749)	0.3007 (0.1334)	0.3241 (0.1535)
	δ = 0.0100	0.2436 (0.0558)	0.2447 (0.1361)	0.2648 (0.1428)	0.3132 (0.0814)	0.2955 (0.1367)	**0.3094** (0.1572)
H = 4	δ = 0.0005	**0.2252** (0.0598)	0.2433 (0.1921)	0.2771 (0.1756)	0.2367 (0.0684)	0.2316 (0.1284)	0.2779 (0.1449)
	δ = 0.0010	0.2268 (0.0581)	**0.2238** (0.1412)	0.2622 (0.1368)	**0.2278** (0.0546)	**0.2311** (0.1201)	**0.2725** (0.1451)
	δ = 0.0050	0.2268 (0.0572)	0.2259 (0.1363)	**0.2539** (0.1333)	0.2629 (0.0853)	0.2516 (0.1459)	0.2943 (0.1575)
	δ = 0.0100	0.2294 (0.0534)	0.2250 (0.1207)	0.2606 (0.1284)	0.2694 (0.0917)	0.2831 (0.1515)	0.3140 (0.1685)
H = 6	δ = 0.0005	0.2206 (0.0637)	0.2424 (0.1875)	0.2568 (0.1544)	0.2229 (0.0595)	0.2281 (0.1102)	0.2727 (0.1455)
	δ = 0.0010	0.2219 (0.0602)	0.2188 (0.1334)	0.2528 (0.1240)	**0.2165** (0.0490)	**0.2264** (0.1143)	**0.2614** (0.1294)
	δ = 0.0050	**0.2102** (0.0557)	**0.2111** (0.1176)	**0.2447** (0.1232)	0.2399 (0.0693)	0.2379 (0.1177)	0.2666 (0.1423)
	δ = 0.0100	0.2221 (0.0501)	0.2225 (0.1189)	0.2470 (0.1280)	0.2658 (0.0784)	0.2488 (0.1263)	0.3021 (0.1554)
H = 8	δ = 0.0005	0.2179 (0.0673)	0.2426 (0.1848)	0.2608 (0.1518)	0.2230 (0.0584)	**0.2211** (0.1052)	0.2682 (0.1482)
	δ = 0.0010	**0.2144** (0.0584)	0.2177 (0.1350)	**0.2491** (0.1121)	**0.2190** (0.0516)	0.2229 (0.1085)	**0.2591** (0.1304)
	δ = 0.0050	0.2221 (0.0552)	0.2256 (0.1350)	0.2617 (0.1277)	0.2281 (0.0615)	0.2331 (0.1084)	0.2728 (0.1512)
	δ = 0.0100	0.2159 (0.0531)	**0.2171** (0.1242)	0.2534 (0.1285)	0.2600 (0.0847)	0.2526 (0.1313)	0.2879 (0.1679)

Table 1 (*ctd.*)

		Summation unit network			*Product unit network*		
		$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$	$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$
$H = 10$	$\delta = 0.0005$	0.2149 (0.0663)	0.2416 (0.1895)	0.2623 (0.1589)	**0.2150** (0.0498)	**0.2199** (0.1079)	0.2551 (0.1310)
	$\delta = 0.0010$	0.2189 (0.0544)	0.2247 (0.1363)	**0.2395** (0.1190)	0.2167 (0.0541)	0.2248 (0.1129)	**0.2528** (0.1304)
	$\delta = 0.0050$	0.2160 (0.0573)	**0.2174** (0.1262)	0.2423 (0.1189)	0.2358 (0.0671)	0.2341 (0.1233)	0.2616 (0.1601)
	$\delta = 0.0100$	**0.2138** (0.0576)	0.2214 (0.1205)	0.2415 (0.1327)	0.2563 (0.0748)	0.2504 (0.1205)	0.2812 (0.1836)
$H = 12$	$\delta = 0.0005$	0.2146 (0.0640)	0.2430 (0.1849)	0.2588 (0.1581)	0.2127 (0.0462)	0.2171 (0.1056)	**0.2552** (0.1416)
	$\delta = 0.0010$	0.2163 (0.0591)	0.2190 (0.1346)	**0.2348** (0.1070)	**0.2110** (0.0460)	**0.2098** (0.1084)	0.2667 (0.1843)
	$\delta = 0.0050$	0.2181 (0.0555)	0.2227 (0.1216)	0.2535 (0.1175)	0.2206 (0.0617)	0.2340 (0.1039)	0.2995 (0.1564)
	$\delta = 0.0100$	**0.2092** (0.0529)	**0.2158** (0.1191)	0.2513 (0.1252)	0.2480 (0.0944)	0.2527 (0.1300)	0.2595 (0.1979)
$H = 14$	$\delta = 0.0005$	0.2139 (0.0626)	0.2395 (0.1867)	0.2537 (0.1565)	**0.2067** (0.0445)	**0.2111** (0.1083)	**0.2514** (0.1390)
	$\delta = 0.0010$	0.2160 (0.0564)	0.2203 (0.1350)	0.2488 (0.1254)	0.2099 (0.0457)	0.2182 (0.1102)	0.2554 (0.1470)
	$\delta = 0.0050$	0.2144 (0.0561)	**0.2153** (0.1187)	**0.2409** (0.1190)	0.2335 (0.0691)	0.2360 (0.1132)	0.2833 (0.1624)
	$\delta = 0.0100$	**0.2120** (0.0547)	0.2254 (0.1158)	0.2483 (0.1233)	0.2391 (0.0774)	0.2477 (0.1178)	0.3008 (0.2717)

Notes: *KLIC*-performance values represent the mean (standard deviation in brackets) of $B = 60$ bootstrap replications differing in both the initial parameter values randomly chosen from [–0.3; 0.3] and the data split. $KLIC(M_1)$: In-sample performance measured in terms of average *KLIC* (the best value for a given H in bold); $KLIC(M_2)$: Validation performance measured in terms of average *KLIC* (the best values for a given H in bold); $KLIC(M_3)$: Out-of-sample performance measured in terms of average *KLIC* (the best values for a given H in bold); M consists of 992 patterns, M_1 of 496 patterns, M_2 of 248 patterns and M_3 of 248 patterns.

It should be emphasised that the main goal of training is to minimise the loss function λ. But it has been observed that forceful training may not produce network models with adequate generalisation ability. We adopted the most common remedy for this problem and checked the model performance in terms of $KLIC(M_2)$ periodically during training to assure that further training improves generalisation, the so-called cross-validation technique.

Alopex is an iterative procedure. In practice, this means that the final results of training may vary as the initial weight settings are changed. Typically, the likelihood functions of feedforward neural network models have many local minima. This implies that the training process is sensitive to its starting point. Despite recent progress in finding the most appropriate parameter initialisation that would help Alopex – but also other iterative procedures – to find near optimal solutions,

the most widely adopted approach still uses random weight initialisation. In our experiments random numbers were generated from [–0.3, 0.3] using the rand_uni function from Press et al. (1992). The order of the input data presentation was kept constant for each run to eliminate its effect on the result.

The Case of Unconstrained Spatial Interactions: Extensive computational experiments with different combinations of H- and δ-values have been performed on DEC Alpha 375 Mhz, with $H \in \{2, 4, 6, 8, 10, 12, 14\}$ and $\delta \in \{0.0005, 0.0010, 0.0025, 0.0050, 0.0100, 0.0250, 0.0500, 0.1000\}$. Selected results of these experiments [$H = 2, 4, 6, 8, 10, 12, 14$ and $\delta = 0.0005, 0.0010, 0.0050, 0.0100$] are reported in Table 1. Training performance is measured in terms of $KLIC(M_1)$, validation performance in terms of $KLIC(M_2)$ and testing performance in terms of $KLIC(M_3)$. The performance values represent the mean of $B = 60$ bootstrap replications, standard deviations are given in brackets.

Some considerations are worth making. *First*, the best result (averaged over the sixty independent simulation runs) in terms of average out-of-sample *KLIC*-performance was obtained with $H = 12$ and $\delta = 0.0010$ in the case of the summation unit neural network model, and with $H = 14$ and $\delta = 0.0005$ in the case of the product unit neural network model. *Second*, there is convincing evidence that the summation unit model outperforms the product unit model version at any given level of model complexity. This is primarily due to the fact that the input data of Ω_L were preprocessed to logarithmically transformed data scaled to [0.1, 0.9]. *Third*, it can be seen that model approximation improves as the complexity of ${}^{\pi}\Omega_L$ grows with increasing H (except H = 12). This appears to be less evident in the case of the summation unit model version.

Fourth, the experiments also suggest that $\delta = 0.0010$ tends to yield the best or at least rather good generalisation performances in both cases of neural network models. The poorest generalisation performance of the summation unit network is obtained for $\delta = 0.0005$ (except H = 8) while $\delta = 0.0100$ leads to the poorest results in the case of the product unit network model (except H = 2 and 12). Fifth, as already mentioned above, forceful training may not produce the network model with the best generalisation ability. This is evidenced for H = 2, 10, 12, 14 in the case of Ω_L, and H = 2, 10, 12 in the case of ${}^{\pi}\Omega_L$. *Finally*, note that the standard deviation illustrates the impact of both, random variations in training, validation and test sets and random variations in parameter initialisations. Most of the variability in prediction performance is clearly coming from sample variation and not from variation in parameter initialisations as illustrated in Fischer and Reismann (2002). This implies that model evaluations based on one specific static split of the data only, the current practice in neural spatial interaction modelling (see, for example, Bergkvist 2000; Reggiani and Tritapepe 2000; Mozolin et al. 2000), have to be considered with great care.

Table 2 summarises the simulation results for the unconstrained neural network models in comparison with the gravity model. Out-of-sample performance is measured in terms of $KLIC(M_3)$. For matters of completeness, also training performance values are displayed. The figures represent averages taken over sixty

independent simulations differing in the bootstrap samples and in the initial parameter values randomly chosen from [–0.3, 0.3].

Table 2 Benchmark comparisons of the summation unit and product unit neural networks, Ω_L and ${}^{\pi}\Omega_L$, with the gravity model τ^{grav} for modelling unconstrained spatial interactions

	Summation unit network [$H = 12$; $\delta = 0.001$]	*Product unit neural network* [$H = 14$; $\delta = 0.0005$]	*Gravity model* [$\delta = 0.0005$]
In-sample (training) performance			
KLIC (M_1)	0.2163 (0.0591)	0.2067 (0.0445)	0.2991 (0.0717)
Out-of-sample (testing) performance			
KLIC (M_3)	0.2348 (0.1070)	0.2514 (0.1390)	0.3036 (0.1952)

Notes: *KLIC*-performance values represent the mean (standard deviation in brackets) of $B = 60$ bootstrap replications differing in the initial parameter values randomly chosen from [+0.3, –0.3] and the data-split; the testing set consists of 248 patterns and the training set of 496 patterns.

If out-of-sample (generalisation) performance is more important than fast learning, then the neural network models exhibit clear and statistically significant superiority. As can be seen by comparing the *KLIC*-values the summation unit neural network model ranks best, followed by the product unit model and the gravity model. The average generalisation performance, measured in terms of $KLIC(M_3)$, is 0.2348 (H = 12), compared to 0.2514 in the case of ${}^{\pi}\Omega_L$ (H = 14), and 0.3036 in the case of τ^{grav}. These differences in performance are statistically significant.[6] If, however, the goal is to minimise execution time and a sacrifice in generalisation accuracy is acceptable, then the gravity model is the model of choice. The gravity model outperforms the neural network models in terms of execution time, the summation unit network model by a factor of 50 and the product unit network model by a factor of 30. But note that this is mainly caused by two factors: first, that our implementations were done on a serial platform even though the neural network models are paralleliseable, and, second, that we implemented a rather time consuming termination criterion ($\kappa = 40,000$) to stop the training process.

The Origin-Constrained Case of Spatial Interactions: Table 3 presents some selected results of experiments with different combinations of $H \in \{2, 4, 6, 8, 10, 12, 14\}$ and $\delta \in \{0.0005, 0.0010, 0.0500, 0.1000\}$. Again some considerations are worth making. *First*, a comparison with Table 1 illustrates that the consideration of a priori information in form of origin constraints clearly improves the generalisation

[6] Assessed by means of the Wilcoxon-Test (comparison of two paired samples). The differences between Ω_L and τ^{grav} are statistically significant at the one percent level (Z = –3.740, Sig. 0.000) as are the differences between ${}^{\pi}\Omega_L$ and τ^{grav} (Z = –3.269, Sig. 0.001). But the differences between Ω_L and ${}^{\pi}\Omega_L$ are not statistically significant at the one percent level (Z = –1.436, Sig. 0.151).

performance more or less dramatically. *Second*, the best result (averaged over the 60 independent simulation runs) in terms of average $KLIC(M_3)$ was achieved with $H = 8$ and $\delta = 0.0500$ in both cases, the origin-constrained summation unit neural network model ${}^{\Sigma}\Omega_1^{OC}$, and the origin-constrained product unit neural network model, ${}^{\pi}\Omega_1^{OC}$. *Third*, the summation unit model version slightly outperforms the product unit version. Again this is primarily due to the logarithmic transformation of the input data in the case of ${}^{\Sigma}\Omega_1^{OC}$.

Fourth, model approximation improves as the complexity of the model grows with increasing H [up to $H = 8$, except $H = 6$ in the case of ${}^{\Sigma}\Omega_1^{OC}$]. *Fifth*, there is clear evidence that $\delta = 0.0500$ tends to lead to the best results (except $H = 6$ in the case of ${}^{\Sigma}\Omega_1^{OC}$), while $\delta = 0.0005$ tends to yield the poorest results, with only two exceptions ($H = 2$ and 4) in the case of ${}^{\Sigma}\Omega_1^{OC}$. *Sixth*, there is strong evidence that the origin-constrained neural network models are much less robust with respect to the choice of the Alopex parameter δ in comparison to their unconstrained counterparts, while the variability in prediction performance over changes in training, internal validation and test samples, and parameter initialisation is lower. *Finally* it is interesting to note that forceful training encourages ${}^{\pi}\Omega_1^{OC}$ to produce the best generalisation ability in all cases considered.

Table 3 The origin-constrained case of spatial interaction – summation unit and product unit neural networks, ${}^{\Sigma}\Omega_1^{OC}$ and ${}^{\pi}\Omega_1^{OC}$ [see Equations (6) and (8)], estimated by the Alopex procedure: The choice of H and δ [T = 1,000; S = 10]

		Summation unit network			*Product unit network*		
		$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$	$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$
$H = 2$	$\delta = 0.0005$	0.3521 (0.0724)	0.3684 (0.0998)	0.3809 (0.1034)	0.2693 (0.0989)	0.2810 (0.1126)	0.2939 (0.1113)
	$\delta = 0.0010$	0.3515 (0.0711)	0.3688 (0.1008)	0.3815 (0.1038)	0.2340 (0.0763)	0.2425 (0.1057)	0.2538 (0.0937)
	$\delta = 0.0500$	**0.1958** (0.0520)	0.2049 (0.0877)	**0.2131** (0.0735)	**0.2037** (0.0575)	**0.2071** (0.0785)	**0.2181** (0.0747)
	$\delta = 0.1000$	0.1982 (0.0518)	**0.1976** (0.0777)	0.2141 (0.0808)	0.2188 (0.0596)	0.2250 (0.0890)	0.2328 (0.0760)
$H = 4$	$\delta = 0.0005$	0.3530 (0.0719)	0.3669 (0.0947)	0.3843 (0.1036)	0.2422 (0.0860)	0.2543 (0.1058)	0.2718 (0.1289)
	$\delta = 0.0010$	0.3540 (0.0710)	0.3655 (0.0949)	0.3855 (0.1042)	0.2175 (0.0809)	0.2265 (0.0857)	0.2404 (0.0905)
	$\delta = 0.0500$	**0.1854** (0.0502)	0.1867 (0.0713)	**0.2032** (0.0747)	**0.1953** (0.0513)	**0.1975** (0.0750)	**0.2125** (0.0749)
	$\delta = 0.1000$	0.1862 (0.0505)	**0.1867** (0.0702)	0.2051 (0.0686)	0.2105 (0.0579)	0.2133 (0.0835)	0.2271 (0.0785)
$H = 6$	$\delta = 0.0005$	0.3523 (0.0723)	0.3695 (0.1002)	0.3820 (0.1070)	0.2351 (0.0749)	0.2462 (0.1013)	0.2662 (0.1166)
	$\delta = 0.0010$	0.3505 (0.0704)	0.3663 (0.1004)	0.3776 (0.1006)	0.2078 (0.0604)	0.2134 (0.0753)	0.2277 (0.0801)

Table 3 (*ctd.*)

		Summation unit network			*Product unit network*		
		$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$	$KLIC(M_1)$	$KLIC(M_2)$	$KLIC(M_3)$
	$\delta = 0.0500$	0.1862 (0.0481)	0.1883 (0.0726)	0.2067 (0.0703)	**0.1915** (0.0474)	**0.1941** (0.0751)	**0.2084** (0.0722)
	$\delta = 0.1000$	**0.1858** (0.0463)	**0.1868** (0.0743)	**0.2050** (0.0701)	0.2019 (0.0468)	0.2046 (0.0764)	0.2211 (0.0784)
$H = 8$	$\delta = 0.0005$	0.3525 (0.0730)	0.3678 (0.1004)	0.3822 (0.1031)	0.2257 (0.0621)	0.2315 (0.0871)	0.2495 (0.0940)
	$\delta = 0.0010$	0.3521 (0.0721)	0.3667 (0.0997)	0.3817 (0.1034)	0.2136 (0.0801)	0.2190 (0.0870)	0.2369 (0.1034)
	$\delta = 0.0500$	**0.1790** (0.0456)	**0.1825** (0.0697)	**0.1989** (0.0684)	**0.1916** (0.0475)	**0.1905** (0.0741)	**0.2076** (0.0707)
	$\delta = 0.1000$	0.1822 (0.0441)	0.1844 (0.0677)	0.2024 (0.0716)	0.2032 (0.0527)	0.2039 (0.0809)	0.2193 (0.0834)
$H = 10$	$\delta = 0.0005$	0.3532 (0.0728)	0.3673 (0.0989)	0.3850 (0.1043)	0.2267 (0.0684)	0.2369 (0.0971)	0.2505 (0.0990)
	$\delta = 0.0010$	0.3516 (0.0713)	0.3675 (0.0986)	0.3818 (0.1037)	0.2014 (0.0488)	0.2076 (0.0697)	0.2219 (0.0755)
	$\delta = 0.0500$	**0.1795** (0.0440)	**0.1820** (0.0671)	**0.2004** (0.0656)	**0.1918** (0.0478)	**0.1949** (0.0755)	**0.2087** (0.0727)
	$\delta = 0.1000$	0.1840 (0.0494)	0.1856 (0.0701)	0.2040 (0.0697)	0.2047 (0.0524)	0.2055 (0.0782)	0.2207 (0.0797)
$H = 12$	$\delta = 0.0005$	0.3586 (0.0821)	0.3720 (0.1038)	0.3877 (0.1108)	0.2357 (0.0895)	0.2443 (0.1009)	0.2637 (0.1428)
	$\delta = 0.0010$	0.3527 (0.0724)	0.3662 (0.0980)	0.3812 (0.1018)	0.2094 (0.0563)	0.2153 (0.0760)	0.2251 (0.0776)
	$\delta = 0.0500$	0.1793 (0.0443)	0.1824 (0.0686)	**0.2005** (0.0647)	**0.1911** (0.0529)	**0.1920** (0.0757)	**0.2101** (0.0717)
	$\delta = 0.1000$	**0.1788** (0.0444)	**0.1822** (0.0670)	0.2012 (0.0691)	0.2038 (0.0532)	0.2081 (0.0845)	0.2202 (0.0763)
$H = 14$	$\delta = 0.0005$	0.3523 (0.0714)	0.3683 (0.1003)	0.3809 (0.1031)	0.2174 (0.0647)	0.2284 (0.0826)	0.2389 (0.0895)
	$\delta = 0.0010$	0.3514 (0.0714)	0.3682 (0.1003)	0.3801 (0.1037)	0.2036 (0.0518)	0.2101 (0.0754)	0.2211 (0.0690)
	$\delta = 0.0500$	**0.1798** (0.0467)	**0.1831** (0.0713)	**0.1992** (0.0663)	**0.1912** (0.0479)	**0.1924** (0.0773)	**0.2126** (0.0725)
	$\delta = 0.1000$	0.1829 (0.0464)	0.1865 (0.0704)	0.2052 (0.0698)	0.2055 (0.0497)	0.2063 (0.0824)	0.2244 (0.0822)

Notes: *KLIC*-performance values represent the mean (standard deviation in brackets) of $B = 60$ bootstrap replications differing in both the initial parameter values randomly chosen from [–0.3; 0.3] and the data-split. $KLIC(M_1)$: In-sample performance measured in terms of average *KLIC* (the best values for a given H in bold); $KLIC(M_2)$: Validation performance measured in terms of average *KLIC* (the best values for a given H in bold); $KLIC(M_3)$: Out-of-sample performance measured in terms of average *KLIC* (the best values for a given H in bold); M consists of 992 patterns, M_1 of 496 patterns, M_2 of 248 patterns and M_3 of 248 patterns.

Table 4 reports the simulation results for the origin-constrained neural network models, ${}^{\Sigma}\Omega_1^{OC}$ and ${}^{\pi}\Omega_1^{OC}$, in comparison with ${}^{orig}\tau^{grav}$. Training and generalisation performance are displayed. The figures represent again averages taken over sixty simulations differing in parameter initialisation and bootstrap samples as in the other tables. The modular summation unit neural network performs best, closely followed by the product unit model version. Both outperform the gravity model predictions.[7] The average out-of-sample performance of ${}^{\Sigma}\Omega_1^{OC}$ with $H = 8$, measured in terms of $KLIC(M_3)$, is 0.1989, compared to 0.2076 in the case of ${}^{\pi}\Omega_1^{OC}$ with $H = 8$, and 0.2726 in the case of ${}^{orig}\tau^{grav}$. The gravity model would be the model of choice if the goal would be to minimise execution time and a sacrifice in generalisation would be acceptable.

Table 4 Benchmark comparisons of the summation unit and product unit neural networks, ${}^{\Sigma}\Omega_1^{OC}$ and ${}^{\pi}\Omega_1^{OC}$, with the gravity model ${}^{orig}\tau^{grav}$ for modelling origin-constrained spatial interactions

	Summation unit network [$H = 8$; $\delta = 0.05$]	*Product unit neural network* [$H = 8$; $\delta = 0.05$]	*Gravity model* [$\delta = 0.1$]
In-sample (training) performance			
KLIC (M_1)	0.1790 (0.0456)	0.1916 (0.0475)	0.2532 (0.0601)
Out-of-sample (testing) performance			
KLIC (M_3)	0.1989 (0.0684)	0.2076 (0.0707)	0.2726 (0.0949)

Notes: *KLIC*-performance values represent the mean (standard deviation in brackets) of $B = 60$ bootstrap replications differing in the initial parameter values randomly chosen from [+0.3, –0.3] and the data-split; the testing set consists of 248 patterns and the training set of 496 patterns.

8 Summary and Directions for Future Research

In this contribution a modest attempt has been made to provide a unified framework for neural spatial interaction modelling including the case of unconstrained and that of origin-constrained spatial interaction flows. We suggested and used a more suitable estimation approach than available in literature, namely maximum likelihood estimation under distributional assumptions of Poisson processes. In this way we could avoid the weakness of least squares and normality assumptions

[7] The differences between ${}^{\Sigma}\Omega_1^{OC}$ and ${}^{orig}\tau^{grav}$ are statistically significant at the one percent level ($Z = -6.684$, Sig. 0.000) as are the differences between ${}^{\pi}\Omega_1^{OC}$ and ${}^{orig}\tau^{grav}$ ($Z = -6.714$, Sig. 0.000), while the differences between the neural network models are not statistically significant at the one percent level ($Z = -2.481$, Sig. 0.130).

that ignore the true integer nature of the flows and approximate a discrete-valued process by an almost certainly misrepresentative continuous distribution. Alopex, a powerful global search procedure, was used to solve the maximum likelihood estimation problem.

Randomness enters in two ways in neural spatial interaction modelling: in the splitting of the data into training, internal validation and test sets on the one side and in choices about parameter initialisation on the other. The paper suggests the bootstrapping pairs approach to overcome this problem as well as the problem of scarce data. In addition one receives a better statistical picture of the variability of the out-of-sample performance of the models. The approach is attractive, but computationally intensive. Each bootstrap iteration requires a run of the Alopex procedure on the training bootstrap set. In very large real world problem contexts this computational burden may become prohibitively large.

Although the discussion has been centered on several general families of neural spatial interaction models, only one of the vast number of neural network architectures and only one – even though powerful – estimation approach were considered. Thus, we emphasise that our results are only a first step towards a more comprehensive methodology for neural spatial interaction modelling. There are numerous important areas for further investigation. Especially desirable is the design of a neural network approach suited to deal with the doubly constrained case. Another area for further research is greater automation of the cross-validation training approach to control maximum model complexity by limiting the number of hidden units. Finding good global optimisation methods for solving the non-convex training problems is still an important area for further research even though some relevant work can be found in Fischer et al. (1999). Finally the model choice problem deserves further research to come up with methods that go beyond the current rules of thumb. We hope that this paper will inspire others to pursue the investigation in neural spatial interaction modelling further as we finally believe that this field is an interesting theoretical area rich with practical applications.

References

Bergkvist E. (2000): Forecasting interregional freight flows by gravity models, *Jahrbuch für Regionalwissenschaft* 20, 133-148

Bergkvist E. and Westin L. (1997): Estimation of gravity models by OLS estimation, NLS estimation, Poisson and neural network specifications, CERUM Regional Dimensions, Working Paper 6

Bia A. (2000): A study of possible improvements to the Alopex training algorithm. In: *Proceedings of the VIth Brazilian Symposium on Neural Networks*, IEEE Computer Society Press, pp. 125-130

Black W.R. (1995): Spatial interaction modelling using artificial neural networks, *Journal of Transport Geography* 3 (3), 159-166

Cover T.M. (1965): Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, *IEEE Transactions on Electronic Computers* 14 (3), 326-334

Efron B. (1982): *The Jackknife, the Bootstrap and Other Resampling Plans.* Philadelphia: Society for Industrial and Applied Mathematics SIAM, Philadelphia

Fischer M.M. and Gopal S. (1994): Artifical neural networks: A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34 (4), 503-527

Fischer M.M. and Leung Y. (eds.) (2001): *GeoComputional Modelling*: *Techniques and Applications*, Springer, Berlin, Heidelberg, New York

Fischer M.M. and Leung Y. (1998): A genetic-algorithms based evolutionary computational neural network for modelling spatial interaction data, *The Annals of Regional Science* 32 (3), 437-458

Fischer M.M. and Reismann M. (2002): Evaluating neural spatial interaction modelling by bootstrapping, *Networks and Spatial Economics* 2 (3), 255-268

Fischer M.M., Reismann M. and Hlavácková-Schindler K. (2001): Neural network modelling of constrained spatial interaction flows, Paper presented at the 41rst Congress of the European Regional Science Association, Zagreb, Croatia [accepted for puplication in the *Journal of Regional Science*]

Fischer M.M., Reismann M. and Hlavácková-Schindler K. (1999): A global search procedure for parameter estimation in neural spatial interacting modelling, *Papers in Regional Science* 78, 119-134

Fotheringham A.S. and O'Kelly M.E. (1989): *Spatial Interaction Models*: *Formulations and Applications*, Kluwer Academic Publishers, Dordrecht, Boston, London

Giles C. and Maxwell T. (1987): Learning, invariance and generalization in high-order neural networks, *Applied Optics* 26 (23), 4972-4978

Gopal S. and Fischer M.M. (1996): Learning in single hidden layer feedforward network models: Backpropagation in a spatial interaction context, *Geographical Analysis* 28 (1), 38-55

Gopal S. and Fischer M.M.(1993): Neural net based interregional telephone traffic models. In: *Proceedings of the International Joint Conference on Neural Networks IJCNN 93 Nagoya, Japan, October 25-29*, pp. 2041-2044

Kullback S. and Leibler R.A. (1951): On information and sufficiency, *Annals of Mathematical Statistics* 22 (1), 78-86

Longley P.A., Brocks S.M., McDonnel R. and Macmillan B. (eds.) (1998): *Geocomputation*: *A Primer*, John Wiley, Chichester [UK], New York

Mozolin M., Thill J.-C. and Usery E.L. (2000): Trip distribution forecasting with multiplayer perceptron neural networks: A critical evaluation, *Transportation Research B* 34, 53-73

Nijkamp P., Reggiani A. and Tritapepe T. (1996): Modelling intra-urban transport flows in Italy, TRACE Discussion Papers TI 96-60/5, Tinbergen Institute, The Netherlands

Openshaw S. (1993): Modelling spatial interaction using a neural net. In: Fischer M.M. and Nijkamp P. (eds.) *Geographic Information Systems, Spatial Modeling, and Policy Evaluation*, Springer, Berlin, Heidelberg, New York, pp. 147-164

Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P. (1992): *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge [MA]

Reggiani A. and Tritapepe T. (2000): Neural networks and logit models applied to commuters mobility in the metropolitan area of Milan. In: Hilmanen V., Nijkamp P. and Reggiani A. (eds.) *Neural Networks in Transport Applications*, Ashgate, Aldershot, pp. 111-129

Rumelhart D.E., Durbin R., Golden R. and Chauvin Y. (1995): Backpropagation: The basic theory. In: Chauvin Y. and Rumelhart D.E. (eds.) *Backpropagation: Theory, Architectures and Applications*, Lawrence Erlbaum Associates, Hillsdale [NJ], pp. 1-34

Thill J.-C. and Mozoli M. (2000): Feedforward neural networks for spatial interaction: Are they trustworthy forecasting tools?" In: Reggiani A. (ed.) *Spatial Economic Science: New Frontiers in Theory and Methodology*, Springer, Berlin, Heidelberg, New York, pp. 355-381

Young T.Y. and Calver T.W. (1974): *Classification, Estimation and Pattern Recognition*, Elsevier, Amsterdam

Figures

PART I Spatial Analysis and GIS

4 GIS and Network Analysis

5 Expert Systems and Artificial Neural Networks for Spatial Analysis and Modelling

PART II Computational Intelligence in Spatial Data Analysis

6 Computational Neural Networks – Tools for Spatial Data Analysis

7 Artificial Neural Networks: A New Approach to Modelling Interregional Telecommunication Flows

8 A Genetic-Algorithms Based Evolutionary Computational Neural Network for Modelling Spatial Interaction Data

PART III GeoComputation in Remote Sensing Environments

9 Evaluation of Neural Pattern Classifiers for a Remote Sensing Application

10 Optimisation in an Error Backpropagation Neural Network Environment

11 Fuzzy ARTMAP – A Neural Classifier for Multispectral Image Classification

PART IV New Frontiers in Neural Spatial Interaction Modelling

12 Neural Network Modelling of Constrained Spatial Interaction Flows

14 A Methodology for Neural Spatial Interaction Modelling

Tables

8 A Genetic-Algorithms Based Evolutionary Computational Neural Network for Modelling Spatial Interaction Data

PART III GeoComputation in Remote Sensing Environments

9 Evaluation of Neural Pattern Classifiers for a Remote Sensing Application

10 Optimisation in an Error Backpropagation Neural Network Environment

11 Fuzzy ARTMAP – A Neural Classifier for Multispectral Image Classification

PART IV New Frontiers in Neural Spatial Interaction Modelling

12 Neural Network Modelling of Constrained Spatial Interaction Flows

13 Learning in Neural Spatial Interaction Models: A Statistical Perspective

14 A Methodology for Neural Spatial Interaction Modelling

Subject Index

Author Index

Acknowledgements

The author and publisher wish to thank the following who have kindly given permission for the use of copyright material.

Chapter 2
Reprinted from *International Encyclopedia of the Social and Behavioral Sciences*, Vol. 22, edited by N.J. Smelser and P.B. Baltes, Spatial analysis in geography, pp. 14752-14758, Copyright © 2001 Elsevier Science Ltd., with permission from Elsevier.

Chapter 3
Reprinted from *Spatial Models and GIS: New Potential and New Models*, edited by M. Wegener and S. Fotheringham, Spatial interaction models and the role of geographic information systems, pp. 33-43, Copyright © 2000 Taylor & Francis, with permission from Taylor & Francis.

Chapter 4
Reprinted from *Handbook of Transport Geography and Spatial Systems*, edited by D. Hensher, K. Button, K.E. Haynes and P. Stopher, GIS and network analysis, pp. 391-408, Copyright © 2004 Elsevier Science Ltd., with permission from Elsevier.

Chapter 5
Reprinted from Fischer M.M. (1994): Expert systems and artificial neural networks for spatial analysis and modelling: Essential components for knowledge-based geographical information systems, *Geographical Systems* 1(3), pp. 221-235, Copyright © 1994 Gordon and Breach Science Publishers SA.

Chapter 6
Reprinted from *GeoComputational Modelling: Techniques and Applications*, edited by M.M. Fischer and Y. Leung, Computational neural networks – Tools for spatial data analysis, pp. 15-34, Copyright © 2001 Springer-Verlag, Berlin, Heidelberg, with permission from Springer-Verlag.

Chapter 7
Reprinted from Fischer M.M. and Gopal S. (1994): Artificial neural networks. A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34(4), pp. 503-527, Copyright © 1994 Blackwell Publishing Ltd., with permission from Blackwell Publishing Ltd.

Chapter 8
Reprinted from Fischer M.M. and Leung Y. (1998): A genetic-algorithms based evolutionary computational neural network for modelling spatial interaction data, *The Annals of Regional Science* 32(3), 1998, pp. 437-458, Copyright © 1998 Springer-Verlag, Berlin, Heidelberg, with permission from Springer-Verlag.

Chapter 9
Reprinted from Fischer M.M., S. Gopal, P. Staufer and K. Steinnocher (1997): Evaluation of neural pattern classifiers for a remote sensing application, *Geographical Systems* 4(2), pp. 221-235, Copyright © 1997 Gordon and Breach Science Publishers SA.

Chapter 10
Reprinted from Fischer M.M. and Gopal S. (1994): Artificial neural networks. A new approach to modelling interregional telecommunication flows, *Journal of Regional Science* 34(4), pp. 503-527, Copyright © 1994 Blackwell Publishing Ltd., with permission from Blackwell Publishing Ltd.

Chapter 11
Reprinted from *Recent Developments in Spatial Analysis – Spatial Statistics, Behavioural Modelling and Computational Intelligence*, edited by Fischer M.M. and Getis A., Fuzzy ARTMAP – A neural classifier for multispectral image classification, Fischer M.M. and S Gopal S., pp. 306-335, Copyright © 1997 Springer-Verlag, Berlin, Heidelberg, with permission from Springer-Verlag.

Chapter 12
Reprinted from Fischer M.M., Reismann M. and Hlavácková-Schindler K. (2003): Neural network modelling of constrained spatial interaction flows: Design, estimation and performance issues, *Journal of Regional Science* 43(1), 2003, pp. 35-61, Copyright © 2003 Blackwell Publishing Ltd., with permission from Blackwell Publishing Ltd.

Chapter 13
Reprinted from Fischer M.M. (2002): Learning in neural spatial interaction models: A statistical perspective, *Journal of Geographical Systems* 4(3), 2002, pp. 287-299, Copyright © 2002 Springer-Verlag, Berlin, Heidelberg, with permission from Springer-Verlag.

Chapter 14
Reprinted from Fischer M.M. and Reismann M. (2002): A methodology for neural spatial interaction modelling, *Geographical Analysis* 34(3), 2002, pp. 207-228, Copyright © 2002 Ohio State University Press, with permission from Blackwell Publishing Ltd.