From Conventional to Neural Spatial Interaction Modelling

Dr. Manfred M. Fischer
Professor and Chair

Department of Economic Geography & Geoinformatics
Vienna University of Economics and Business Administration
A-1090 Vienna
Rossauer Lände 23/1
phone: +43-1-31336/4836
fax: +43-1-31336/703
e-mail: Manfred.Fischer@wu-wien.ac.at

Workshop on "Spatial Aspects of Demography", MPI for Demographic Research, Rostock, Germany, May 16-17, 2001
From Conventional to Neural Spatial Interaction Modelling

1 Introduction

One of the major intellectual achievements and, at the same time, perhaps the most useful contribution of spatial analysis to social science literature is the development of spatial interaction models. Spatial interaction can be broadly defined as movement of people, commodities, capital and/or 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 (e.g. telephone calls), airline passenger 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/her possible destination. It is the pervasiveness of this type of trade-off in spatial behaviour which has made spatial interaction analysis and modelling so important and the subject of intensive investigation in human geography and regional science (Fotheringham and O'Kelly 1989).

Since the pioneering work of Wilson (1970) on entropy maximization, however, there has been surprisingly little innovation in the design of spatial interaction models. The principal exceptions include the competing destinations version of Fotheringham (1983), the use of genetic algorithms to breed new forms of spatial interaction models, either directly (Openshaw 1988) or by genetic programming (Turton, Openshaw and Diplock 1997), and the design of neural spatial interaction models (Fischer and Gopal 1994a, b, Gopal and Fischer 1993, Openshaw 1993). Neural spatial models are termed neural in the sense that they are based on neural computational approaches, inspired by neuroscience. They are more closely related to spatial interaction models of the gravity type, and under commonly met conditions they can be viewed as a special class of general feedforward neural network models with a single hidden layer and sigmoidal transfer functions (see Fischer 1998b). Rigorous mathematical proofs for the universality of such feedforward neural network models (see, among others, Hornik, Stinchcombe and
2 The Model Class under Consideration

Suppose we are interested in approximating a $N$-dimensional spatial interaction function $F: \mathbb{R}^N \rightarrow \mathbb{R}$, where $\mathbb{R}^N$ as $N$-dimensional Euclidean real space is the input space and $\mathbb{R}$ as 1-dimensional Euclidean real space is the output space. This function should estimate spatial interaction flows from regions of origin to regions of destination. The function $F$ is not explicitly known, but given by a finite set of samples $S=\{(x_k, y_k), k=1, ..., K\}$ so that $F(x_k)=y_k$, $k=1, ..., K$. The set $S$ is the set of pairs of input and output vectors. The task is to find a continuous function that
approximates set $S$. In real world applications, $K$ is a small number and the samples contain noise.

To approximate $F$, we consider the class of neural spatial interaction models $\Omega$ with one hidden layer, $N$ input units, $J$ hidden units and one output unit. $\Omega$ consists of a composition of transfer functions so that the (single) output of $\Omega$ is

$$y = \Omega(x, w) = \psi\left(\sum_{j=0}^{J} w_j \phi\left(\sum_{n=0}^{N} w_{jn} x_n \right)\right)$$

(1)

The expression $\Omega(x, w)$ is a convenient short-hand notation for the model output since this depends only on inputs and weights. Vector $x = (x_1, \ldots, x_N)$ is the input vector augmented with a bias signal $x_0$ which can be thought as being generated by a 'dummy unit' whose output is clamped at 1. The $w_j$'s represent input to hidden connection weights and the $w_j$'s hidden to output weights (including the bias). The symbol $w$ is a shorthand notation of the $d=(J(N+1)+J+1)$-dimensional vector of all the $w_j$ and $w$ network weights and biases (i.e. model parameters). $\phi(.)$ and $\psi(.)$ are differentiable non-linear transfer functions of the hidden units $j = 1, \ldots, J$ and the output unit, respectively.

The architecture of the hidden layer feedforward network model determines the precise form of the function $\Omega$. Different network architectures will lead to different forms for $\Omega$. Each neural spatial interaction model $\Omega(x, w)$ can be represented in terms of a network diagram (see Fig. 1) such that there is a one-to-one correspondence between components of $\Omega$ 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 with $N=3$. 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 $\psi$ and $\phi$ ($j=1, \ldots, J$).

Different types of transfer functions $\phi$ ($j=1, \ldots, J$) and $\psi$ will lead to different particular computational networks. If the transfer functions are taken to be linear, functional form (1) becomes a special case of the general linear regression model. The crucial difference is that here we consider the weight parameters appearing in the hidden and output layers as being adaptive so that their values can be changed during the process of network training [in statistical terminology: parameter estimation].

The novelty and fundamental contribution of the feedforward neural network approach to spatial interaction analysis derives from its focus on functions such as $\phi$ and $\psi$ in (1), and much less on the associated learning methods which will be discussed in Section 4. Equation (1) is a rather general class of spatial interaction functions, something which took a long time to be appreciated. Hornik, Stinchcombe and White (1989) and other authors have shown that neural networks with a single hidden layer such as (1) can approximate any continuous function $\Omega$ uniformly on compacta under generally met conditions, by increasing the size of $J$ of the hidden layer. Because of this universal approximation property, single hidden layer feedforward network models are useful for applications in
spatial analysis in general and spatial interaction analysis in particular, and a host of related tasks. There are also some results on the rate of approximation [i.e. how many hidden units are needed to approximate to a specified accuracy], but as always with such results they are no guide to how many hidden units might be needed in any practical problem.

![Diagram of neural spatial interaction models](image)

**Fig. 1** Representation of the general class of neural spatial interaction models defined by equation (1) for \( N=3 \) [biases not shown]

Neural spatial interaction modelling may be viewed as a *three-stage process* as outlined in Fischer and Gopal (1994a):

- The *first stage* refers to the identification of a model candidate from the class (1) of neural spatial interaction models [i.e. the model choice task]. This involves both the specification of appropriate transfer functions \( \psi \) and \( \phi_j \) (\( j=1, ..., J \)) and the determination of an adequate network topology of \( \Omega \).

- The *second stage* involves the network training problem [parameter estimation problem], that is the determination of an optimal set of model parameters where optimality is defined in terms of an error or performance function.
• The third stage is concerned with testing and evaluating the out-of-sample [generalization] performance of the chosen model.

No doubt, the tasks of model selection and parameter estimation are of crucial importance for the success of real world neural spatial interaction modelling. Obviously, these two processes are intertwined. If an appropriate \( J \) and 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.

Following Fischer and Gopal (1994a, b), it seems useful to consider the case \( N=3 \), i.e. the input space will be a closed interval of the three-dimensional Euclidean space \( \mathbb{R}^3 \). The three input units are chosen to correspond to the independent variables of the classical unconstrained spatial interaction model. They represent measures of origin propulsiveness, destination attractiveness and spatial separation. The output unit corresponds to the dependent variable of the classical model and represents the spatial interaction flows from origin to destination. The above general framework for neural spatial interaction analysis has been used for many years. Experience has found the logistic function to be an appropriate choice for \( \psi \) and \( \phi_j \) (\( j = 1, \ldots, J \)). This form of specification of the transfer functions leads to the following special class \( \Omega (x, w) \) of spatial interaction functions

\[
\Omega (x, w) = \left\{ 1 + \exp \left[ - \sum_{j=1}^{J} w_j \left( 1 + \exp \left( - \sum_{n=0}^{N} w_{jn} x_n \right) \right)^{-1} \right] \right\}^{-1}
\]

with \( N=3 \). Such models may be viewed as non-linear regression models of a quite specific form. If is \( \psi \) taken to be the identity function we arrive at

\[
\Omega (x, w) = \sum_{j=1}^{J} w_j \left( 1 + \exp \left( - \sum_{n=0}^{N} w_{jn} x_n \right) \right)^{-1}
\]

A heuristic reason why neural spatial interaction models of type (2) or (3) might work well with modest numbers of hidden units is that the first stage allows a projection onto a subspace of \( \mathbb{R}^N \) within which the approximation can be performed. In this these neural models share many of the properties of projection pursuit regression. Indeed, for theoretical purposes (3) is essentially equivalent to projection pursuit regression (see Friedman and Stuetzle 1981).

Since the above neural network models can perform essentially arbitrary non-linear functional mappings, a neural spatial interaction model could - in principle - be used to map the raw input data directly onto the required final output values. In practice, for all but the simplest problems, such an approach will generally give poor results for a number of reasons. For most applications it is necessary first to transform the data into some new representation before training a model. In the simplest case, pre-processing may take the form of a linear transformation of the
input data, and possibly also of the output data [post-processing]. More complex pre-processing may involve reduction of the dimensionality of the input data.

Another important way in which model performance can be improved is through the incorporation of prior knowledge which refers to relevant information [such as production or attraction constraints in the spatial interaction system] that might be used to develop a solution and which is additional to that provided by the training data. Prior knowledge can either be incorporated into the network structure itself or into the pre-processing and post-processing stages. It can also be used to modify the training process through the use of regularisation as discussed in Section 5.

3 Training Neural Spatial Interaction Models: Classical Techniques

In this section we restrict our scope to the network training problem and to training algorithms that train a fixed member of the class $\Omega$ of spatial interaction models. The approximation of $\Omega$ to $F$ then only depends on the learning samples, and the learning (training) algorithm that determines the parameter $w$ from $S$ and the model specification.

The Parameter Estimation Problem

Let us assume the hidden transfer function $\phi(.)$ to be identical, $\phi(.)=\psi(.)$ for all $j=1, \ldots, J$, and the logistic function, and moreover the model to have a fixed topology, ie. $N=3$, and $J$ being predetermined. Then, the goal of learning is to find suitable values $w^*$ for the network weights of the model such that the underlying mapping $F: R^3 \rightarrow R$ represented by the training set of samples is approximated or learned. The process of determining optimal parameter values is called training or learning and can be formulated in terms of minimization of an appropriate error function (or cost function) $E$ to measure the degree of approximation with respect to the actual setting of network weights. The most common error function is the squared-error function of the patterns over the finite set of training data, so that the parameter estimation problem may be defined for batch learning as the following minimization problem:

$$\min_{w} E(w, S) = \min_{w} \sum_{(x, y) \in S} [y_i - \Omega(x_i, w)]^2$$

where the minimization parameter is the weight vector $w$ defining the search space. The function $E(w, S)$ is non-negative continuously differentiable on the $d$-dimensional parameter space which is a finite dimensional closed bounded
domain and, thus, compact, and so \( E(w, S) \) assumes its minimum value \( w^* \) as the weight minimum. Characteristically, there exist many minima all of which satisfy

\[
\nabla E(w, S) = 0
\]

(5)

where \( \nabla E(w, S) \) denotes the gradient error function in the \( d \)-dimensional parameter space. The minimum for which the value of \( E(w, S) \) 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. Due to the non-linearity of the transfer functions it is not possible to find closed-form solutions for this optimization problem.

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 patterns \( s_k \) 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 \( K' (1<K'\leq K) \). Both, the on-line and epoch-based (\( K' \) small) versions are not consistent with optimization 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, Jost and Werner 1993, and Fischer and Staufer 1999 in the context of a spectral pattern classification problem).

The Optimization Strategy

Above the network training problem has been formulated as a problem of minimizing the least square error function, and, thus, as a special case of function approximation where no explicit model of the data is assumed. Most of the optimization procedures used to minimize functions are based on the same strategy. The minimization is a local iterative process in which an approximation to the function in a neighbourhood of the current point in parameter space is minimized. The approximation is often given by a first- 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:

1. choose an initial vector \( w \) in parameter space and set \( \tau = 1 \),
2. determine a search direction \( d(\tau) \) and a step size \( \eta(\tau) \) so that
   \[
   E(w(\tau) + \eta(\tau) d(\tau)) < E(w(\tau)) \quad \tau = 1, 2, ... \]  
   (6)
3. update the parameter vector
\[ w(\tau + 1) = w(\tau) + \eta(\tau) \, d(\tau) \quad \tau = 1, 2, ... \] (7)

(iv) if \( dE(w)/dw \neq 0 \) then set \( \tau = \tau + 1 \) and go to (ii), else return \( w(\tau + 1) \) 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 \( w_k(\tau) \) is updated after every presentation of \( s_k = (x_k, y_k) \). 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 \( d(\tau) \) 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 with respect to the network parameters, and, second, the computation of the parameter \( \eta(\tau) \) and the direction vector \( d(\tau) \) based upon these derivatives. The evaluation of the derivatives of the error function is most commonly performed by the backpropagation technique which provides a computationally efficient procedure for calculating such derivatives (Rumelhart, Hinton and Williams 1986).

**Optimization Techniques for Parameter Adjustments**

In numerical optimization different techniques for the computation of the parameter \( \eta(\tau) \) and the direction vector \( d(\tau) \) are known (see, for example, Luenberger 1984, Fletcher 1986). In particular, three techniques gained popularity:

(i) The **steepest-descent** (gradient) methods (GD) calculate the actual search direction \( d(\tau) \) as the negative gradient

\[ d(\tau) := -\nabla E(w(\tau)) \quad \tau = 1, 2, ... \] (8)

(ii) **Conjugate gradient** (CG) methods calculate the actual search direction \( d(\tau) \) 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-Ribiere variant of conjugate gradient procedures (see Press et al. 1992), the search direction is computed as

\[ d(\tau) := -\nabla E(w(\tau)) + \beta(\tau) \, d(\tau-1) \quad \tau = 1, 2, ... \] (9)

with
\( d(0) = -\nabla E(w(0)) \) \hspace{1cm} (10)

where \( \beta(\tau) \) is a scalar parameter that ensures that the sequence of vectors \( d(\tau) \) satisfying the following condition expressed as

\[
\beta(\tau) = \frac{[\nabla E(w(\tau)) - \nabla E(w(\tau - 1))]'}{\nabla E(w(\tau - 1))} \hspace{1cm} (11)
\]

\( w(\tau - 1)' \) is the transpose of \( w(\tau - 1) \). Note that the CG algorithm utilizes information about the direction search \( d(\tau - 1) \) 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 \( \tau \) 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

\[
d(\tau) = -H(\tau) \nabla E(w(\tau)) \hspace{1cm} (12)
\]

where \( H(\tau) \) is some \( w \times w \) symmetric positive definite matrix and denotes the current approximation to the inverse of the Hessian matrix, i.e.,

\[
H(\tau) \equiv [\nabla^2 E(w(\tau))]' \hspace{1cm} (13)
\]

where

\[
H(\tau) = \left\{ I - \frac{d(\tau - 1)(g(\tau - 1))'}{d(\tau - 1) g(\tau - 1)} \right\} H(\tau - 1) \left\{ I - \frac{g(\tau - 1)(d(\tau - 1))'}{d(\tau - 1) g(\tau - 1)} \right\} + \frac{d(\tau - 1)(d(\tau - 1))'}{d(\tau - 1) d(\tau - 1)} \hspace{1cm} (14)
\]

with

\[
g(\tau - 1) := -\nabla E(w(\tau)) - \nabla E(w(\tau - 1)). \hspace{1cm} (15)
\]
\( H \) is initialised usually with the identity matrix \( I \) 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 error function \( E \) or its derivative, both of which raise the complexity. Characteristically, the parameter \( \eta \) is chosen to minimize

\[
E(\tau) = E(\mathbf{w}(\tau)) + \eta \mathbf{d}(\tau)
\]  

(16)

in the \( \tau \)-th iteration. This gives an automatic procedure for setting the step length, once the search direction \( \mathbf{d}(\tau) \) 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 Fischer and Staufer (1999). 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 \( \eta \) by trial and error. Inefficiency is primarily due to the fact that the minimization 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 quickprop (Fahlmann 1988).

In contrast, the conjugate gradient and quasi-Newton procedures are intrinsically off-line parameter adjustment techniques, and evidently more sophisticated optimization 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 \( \mathbf{w}\times\mathbf{w} \)-dimensional matrices \( H(\tau) \) need to be generated as in the quasi-Newton procedures. The search direction is chosen by appropriately setting the \( \beta \) so that \( \mathbf{d} \) distorts as little as possible the minimization achieved by the previous search step. A major difficulty is that for a non-quadratic error function the obtained directions are \textit{not} necessarily descent directions and numerical instability can result (Battiti and Tecchiolli 1994). Periodically, it might be necessary to restart the optimization process by a search in the steepest descent direction when a non-descent search direction is generated. It is worthwhile to mention that the gradient descent procedures can be viewed as a form of gradient descent with an adaptive momentum \( \beta(\tau) \), the important difference being that \( \eta(\tau) \) and \( \beta(\tau) \) in conjugate
gradient are automatically determined at each iteration (see equations (11) and (16)).

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 \( w \times w \)-dimensional matrices \( H(\tau) \), especially in the case of large-sized problems (Shanno 1990). In quasi-Newton procedures, the matrices \( H(\tau) \) are positive definite approximations of the inverse Hessian matrix obtained from gradient 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 utilizing 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 \( d(\tau) \) is always a descent search direction. They are today the most efficient and sophisticated optimization techniques for batch training. But they are expensive both in computation and memory. Large-sized real-world problems implying larger \( w \) could lead to prohibitive memory requirements (Shanno 1990).

In general, the above minimization techniques find the local minima efficiently and work best in unimodal problems. They show difficulties when the surface of the parameter space is flat (i.e. gradients close to zero), when gradients can be in a large range, or when the surface is very rugged. When gradients vary greatly, the search may progress too slowly, when the gradient is small, and may overshoot where the gradient is large. When the error surface is rugged, a local search from a random starting point generally converges to a local minimum close to the initial point and a worse solution than the global minimum.

4 A New Global Search Approach for Network Training: The Differential Evolution Method

Global search algorithms employ heuristics to allow to escape from local minima. These algorithms can be classified as probabilistic or deterministic. Of the few deterministic global minimization methods developed, most apply deterministic heuristics to bring search out of a local minimum. Other methods, like covering methods, recursively partition the search space into subspaces before searching. None of these methods operate well or provide adequate coverage when the search space is large as it is usually the case in neural spatial interaction modelling.
Probabilistic global minimization methods rely on probability to make decisions. The simplest probabilistic algorithm uses restarts to bring search out of a local minimum when little improvement can be made locally. More advanced methods rely on probability to indicate whether a search should ascend from a local minimum: simulated annealing, for example, when it accepts uphill movements. Other probabilistic algorithms rely on probability to decide which intermediate points to interpolate as new trial parameter vectors: random recombinations and mutations in evolutionary algorithms (see, for example, Fischer and Leung 1998).

Central to global search procedures is a strategy that generates variations of the parameter vectors. Once a variation is generated, a decision has to be made whether or not to accept the newly derived trial parameter. Standard direct search methods (with few exceptions such as simulated annealing) utilize the greedy criterion to make the decision. Under this criterion, a new parameter vector is accepted if and only if it reduces the value of the error function. Although this decision process converges relatively fast, it has the risk of entrapment in a local minimum. Some stochastic search algorithms like genetic algorithms, and evolution strategies employ a multipoint search strategy, in order to escape from local minima.

The Differential Evolution Method (DEM), originally developed by Storn and Price (1996, 1997) and adopted by Fischer, Hlaváčková-Schindler and Reismann (1999) for spatial interaction analysis, is a global optimization algorithm that employs a structured, yet randomized parallel multipoint search strategy which is biased towards reinforcing search points at which the error function \( E(w, S) \) being minimized has relatively low values. The DEM is similar to simulated annealing in that it employs a random (probabilistic) strategy. But one of the apparent distinguishing features of DEM is its effective implementation of parallel multipoint search. DEM maintains a collection of samples from the search space rather than a single point. This collection of samples is called population of trial solutions.

To start the stochastic multipoint search, an initial population \( P \) of, say \( M \), \( d \)-dimensional parameter vectors \( P(0) = \{w_0(0),...,w_M(0)\} \) is created. For the parameter estimation problem at hand \( d = J(N+1)+J+1 \). Usually this initial population is created randomly because it is not known a priori, where the globally optimal parameter is likely to be found in the parameter space. If such information is given, it may be used to bias the initial population towards the most promising regions of the search space by adding normally distributed random deviations to this a priori given solution candidate. From this initial population, subsequent populations \( P(1), P(2),...,P(\tau),... \) will be computed by a scheme that generates new parameter vectors by adding the weighted difference of two vectors to a third. If the resulting vector yields a lower error function value than a predetermined population member, the newly generated vector will replace the vector which it was compared to, otherwise the old vector is retained. Similarly to evolution strategies, the greedy criterion is used in the iteration process and the probability distribution functions determining vector mutations are not a priori given. The scheme for generating \( P(\tau + 1) \) from \( P(\tau) \) with \( \tau \geq 0 \) may be summarized by three major stages: the construction of \( v(\tau + 1) \)-vectors from vector \( w(\tau) \)-vectors
(Stage 1), the construction of \( u(\tau + 1) \)-vectors from \( v(\tau + 1) \) vectors and \( w(\tau) \)-vectors (Stage 2), and the decision criterion whether or not the \( u(\tau + 1) \) - vector should become members of the population (Stage 3) representing possible solutions of the parameter estimation problem under consideration at step \( \tau + 1 \) of the iteration process. The iteration process continues until some stopping criterion applies.

**Stage 1:** For each population member \( w_{m\tau}, m = 0, 1, ..., M - 1 \), a perturbed vector \( v_{m}(\tau + 1) \) is generated according to:

\[
v_{m}(\tau + 1) = w_{m\tau} + \kappa(w_{r_1\tau} - w_{r_2\tau}) \tag{17}
\]

with \( r_1, r_2 \) integers chosen randomly from \{0,..., M - 1\} and mutually different. The integers are also different from the running index \( m \). \( \kappa \in (0, 2] \) is a real constant factor which controls the amplification of the differential variation \((w_{r_1\tau} - w_{r_2\tau})\). The parameter vector \( w_{m\tau} \) which is perturbed to yield \( v_{m}(\tau + 1) \) is the best parameter vector of population \( P(\tau) \).

**Stage 2:** In order to increase the diversity of the new parameter vectors, some specific type of crossover may be introduced. One might use the crossover of the exponential type (Storn and Price 1996), yielding the vector:

\[
u_{m}(\tau + 1) = (u_{0m}(\tau + 1), ..., u_{d-1m}(\tau + 1)) \tag{18}
\]

where

\[
u_{m}(\tau + 1) = \begin{cases} v_{m}(\tau + 1) & \text{for } k = \langle i \rangle_d, ..., \langle i + R - 1 \rangle_d \\ w_{m\tau} & \text{for all other } k \in [0, d - 1] \end{cases} \tag{19}
\]

is formed. The brackets \( \langle \cdot \rangle_d \) denote the modulo function with modulus \( d \). In other words, a sequence of \( R \) coordinates of vector \( u(\tau + 1) \) is identical to the corresponding coordinates of vector \( v(\tau + 1) \), whereas the other coordinates of \( u(\tau + 1) \) are retained as the original values of \( w(\tau) \). The starting index \( i \) in (19) is a randomly chosen integer from the interval \([0, d - 1]\). The integer \( R \), which denotes the number of parameters that are going to be exchanged, is drawn from the interval \([1, d]\) with the probability \( Pr(R = \eta) = (CR)^\eta \), where \( \eta > 0 \) and \( CR \in [0, 1] \) is the crossover probability and forms a control variable for the scheme. The random decisions for both \( i \) and \( R \) are made anew for each newly generated trial vector \( u_{m}(\tau + 1) \). It is worth noting that \( CR = 1 \) implies \( u_{m}(\tau + 1) = v_{m}(\tau + 1) \).
Stage 3: The decision whether or not $u_m(\tau+1)$ should become a member of $P(\tau+1)$, is based on the greedy criterion. If:

$$E(u_m(\tau+1)) < E(w_m(\tau))$$

(20)

then $w_m(\tau)$ is replaced by $u_m(\tau+1)$ otherwise the old value $w_m(\tau)$ is retained as $w_m(\tau+1)$.

When using non-linear optimization algorithms such as DEM, some choice must be made when to stop the training process. Possible choices are listed below:

(i) Stop after a fixed number of iterations. The problem with this approach is that it is difficult to know a priori how many iterations would be appropriate. But an approximate idea can be obtained from some preliminary tests.

(ii) Stop when the error function falls below some specified value. This criterion suffers from the problem that the a priori specified value may never be reached so a limit on iterations as in (i) is also required.

(iii) Stop when the relative change in error function falls below some a priori specified value. This may lead to premature termination if the error function decreases relatively slowly during some part of the training process.

(iv) Stop training when the error measured using an independent validation set starts to increase. This approach, called early stopping or cross-validation, may be used as part of a strategy to optimize the generalization performance of the network model (see Fischer and Gopal 1994a for details).

In practice, some combination of the above strategies may be employed as part of a largely empirical process of parameter estimation.

Taking Austrian interregional telecommunication traffic data as testbed Fischer, Hlaváčková-Schindler and Reismann (1999) have found that the differential evolution method slightly outperforms backpropagation of conjugate gradients in terms of in-sample and out-of-sample performance, but at a very high price of computational costs. The issue of accuracy has ramifications with respect to a priori knowledge of the response surface. If a correct neural spatial interaction model structure is assumed, DEM, in general, tends to be slower than conventional local optimization schemes such as the conjugate gradient procedure used as benchmark. This results from the inefficiency of not using information about the gradient of the error function although gradient methods could be incorporated in parallel with differential evolution. There is potential for further development of this novel strategy in this direction.
5 Selecting Neural Spatial Interaction Models: The Model Choice Issue

One of the central issues in neural network modelling in general and neural spatial interaction modelling in particular is that of model selection. The model specification problem includes, first, the specification of appropriate transfer functions $\psi$ and $\phi$ ($j=1, \ldots, J$) and, second, the determination of an adequate network topology of $\Omega$. In the case of the model classes $\Omega_L$ and $\Omega_L^*$ this essentially means the determination of the optimum size of $J$, because an increase in $J$ - or, in other words, in the number of connection parameters - will generally allow a smaller value of the error to be found. The goal of model selection is to optimize the complexity of the model in order to achieve the best out-of-sample performance, that is, generalization (see Fischer 2000).

The Concept of the Bias-Variance Trade-Off

Considerable insight into this phenomenon can be achieved by introducing the concept of the bias-variance trade-off, in which the generalization error is decomposed into the sum of the bias squared plus the variance. Following Bishop (1995) the sum-of-squares error, in the limit of an infinite data set $S$, can be written in the form of

$$
E(w, S) = \frac{1}{2} \int (\Omega(x, w) - \langle y|x \rangle)^2 p(x) \, dx + \frac{1}{2} \int (\langle y^2|x \rangle - \langle y|x \rangle^2) p(x) \, dx
$$

(21)

in which $p(x)$ is the unconditional density of the input data, $\langle y|x \rangle$ denotes the conditional average, or regression, of the target data $y$ given by

$$
\langle y|x \rangle = \int y \, p(y|x) \, dy
$$

(22)

where $p(y|x)$ is the conditional density of the target variable $y$ conditioned on the output vector $x$. Similarly

$$
\langle y^2|x \rangle = \int y^2 \, p(y|x) \, dy.
$$

(23)

Note that the second term in (21) is independent of the spatial interaction function $\Omega(x, w)$ and, thus, is independent of the network weights $w$. The optimal model, in the sense of minimizing the sum-of-squares error is the one that makes the first term in (21) vanish, and is given by $\Omega(x, w) = \langle y|x \rangle$. The second term represents the intrinsic noise in the data and sets a lower limit on the error that can be achieved.
In real world application contexts we have to deal with the problems arising from a finite size data set. Suppose, we consider a training set $S$ consisting of $K$ patterns that we utilize to determine the neural spatial interaction function $\Omega(x, w)$. Now consider a whole ensemble of possible data sets, each containing $K$ patterns, and each taken from the same fixed joint distribution $p(x, y)$. The optimal network model is given by the conditional average $\langle y|x \rangle$. A measure of how close the actual spatial interaction function $\Omega(x, w)$ is to the desired one is given by the integrand of the first term in (21): $(\Omega(x, w)-\langle y|x \rangle)^2$. The value of this quantity will depend on the particular data set $S$ on which it is trained. We can eliminate this dependence by considering an average over the complete ensemble of data sets, that is

$$\epsilon_s[(\Omega(x, w)-\langle y|x \rangle)^2]$$

where $\epsilon_s(.)$ denotes the expectation [ensemble average]. If model $\Omega$ was always a perfect predictor of the regression function $\langle y|x \rangle$ then this error would be zero. A non-zero error can arise essentially due to two distinct reasons: First, it may be that the model $\Omega$ is on average different from the regression function. This is termed bias. Second, it may be that $\Omega$ is very sensitive to the particular data set $S$, so that, a given $x$, it is larger than the required value for some data sets, and smaller for other data sets. This is called variance. We can make the decomposition into bias and variance explicit by writing (24) in a somewhat different, but mathematically equivalent form (see Bishop 1995):

$$\epsilon_s[(\Omega(x, w)-\langle y|x \rangle)^2] = \epsilon_s((\Omega(x, w)-\langle y|x \rangle)^2) + \epsilon_s((\Omega(x, w)-\epsilon_s((\Omega(x, w))))^2)$$

where the first term of the right hand side of the equation denotes the bias squared and the second term the variance. The bias measures the extent to which the average over all data sets of the spatial interaction function differs from the desired function $\langle y|x \rangle$. Conversely, the variance measures the extent to which $\Omega$ is sensitive to the particular choice of data sets.

A neural spatial interaction model that is too simple [i.e. small $J$], or too inflexible, will have a large bias and smooth out some of the underlying structure in the data [corresponding to high bias], while one that has too much flexibility in relation to the particular data set will overfit the data [corresponding to high variance] and have a large variance. In either case, the performance of the network on new data [i.e. generalization performance] will be poor. This highlights the need to optimize the complexity in the model selection process in order to achieve the best generalization.
Model Selection Techniques

Both the theoretical and practical side of the model selection problem has been intensively studied in the field of neural networks and a vast array of methods have been suggested to perform this task. Most approaches view model selection as a process consisting of a series of steps that are performed independently.

**Step 1:** The first step consists of choosing a specific parametric representation [i.e. model] that is oversized in comparison to the size of the training set used.

**Step 2:** Then in the second step either an error function such as $E$ [possibly including a regularization term] is chosen directly, or in a Bayesian setting, prior distributions on the elements of the data generation process (noise, model parameter, regularizers, etc.) are specified from which an objective function is derived.

**Step 3:** Utilizing the error function specified in Step 2, the training process is started and continued until a convergence criterion is fulfilled. The resulting parametrization of the given model architecture is then placed in a pool of model candidates from which the final model will be chosen.

**Step 4:** To avoid overfitting, model complexity must be limited. Thus, the next step usually consists of modifying the network model architecture [for example, by pruning weights], or of the penalty term [for example, by changing its weighting in the objective function], or of the Bayesian prior distributions. The last two modifications then lead to a modification of the objective function. It is worthwhile noting that this establishes a new framework for the training process that is then restarted and continued until convergence, yielding another model for the pool.

This process is iterated until the model builder is satisfied that the pool contains a reasonable diversity of model candidates, that are then compared with each other using some estimator of generalization ability, for example, the average relative variance [i.e. a normalized mean squared error metric] given by (Fischer and Gopal 1994a)

$$\text{ARV} \left( S \right) = \frac{\sum_{(x_k, y_k) \in S} (y_k - \Omega \left( x_k, w \right))^2}{\sum_{(x_k, y_k) \in S} (y_k - \bar{y})^2} \quad (26)$$
where $y$ denotes the target vector and $\bar{y}$ the average over $K$ desired values in $S$.

The methods employed for training may be very sophisticated (see, for example, Fischer and Stauffer 1999). In contrast to this, the choice and modification of the network model architecture and objective function is generally ad hoc, or is directed by a search heuristic (see, for example, Openshaw 1993). In this section three principal approaches for directing model modification and selection are considered. The first approach is by use of pruning techniques. The principal idea of pruning is to reduce the number of model parameters by removing dispensable ones. Thus, pruning techniques function by training an oversized neural network model with a fixed, but larger $J$ to a minimum of $E(w, S)$, then testing elements of the model, such as connection parameters, for relevance. Those elements with poor test results are then deleted and the modified network model is retrained. In this approach one uses the information in an existing model to direct the search to the best 'neighbouring' model.

Clearly, various choices have to be made utilizing this approach. The most important is how to decide which parameter weights should be removed. The decision is generally based on some measure of the relative importance, or saliency, of different weight parameters. The simplest concept of saliency is to suppose that small weights are less important than large weights, and to use the absolute magnitude of a parameter value as a measure of its importance, under the assumption that the training process naturally forces non-relevant weights into a region around zero.

A major shortcoming of this pruning technique is its weak theoretical motivation. Since parameter estimation is defined in terms of the minimization of the error function $E$, it is natural to use the same error function to find a more principled definition of saliency. Especially, we could define the saliency of a model parameter as the change in the error function that results from deletion of that parameter. This could be implemented by direct evaluation so that, for each parameter in the trained network model in turn, the parameter is temporarily set to zero and the error function re-evaluated. Though conceptually attractive, such an approach will be computationally demanding in the case of larger neural spatial interaction models.

Consider instead the change in the error function due to small changes in the parameter values (Le Cun, Denker and Solla 1990). If the parameter $w_i$ is changed to $w_i + \delta w_i$, then the corresponding change in the error function $E$ is given by

$$
\delta E(w, S) = \sum_i \frac{\partial E(w, S)}{\partial w_i} \delta w_i + \frac{1}{2} \sum_i \sum_j H_{ij} \delta w_i \delta w_j + O(\delta w^3)
$$

(27)

where $H_{ij}$ denote the elements of the Hessian matrix

$$
H_{ij} = \frac{\partial^2 E(w, S)}{\partial w_i \partial w_j} = \nabla^2 E(w, S).
$$

(28)
If we assume that the parameter estimation process has converged, then the first term in (27) will vanish. Le Cun, Denker and Solla (1990) approximate the Hessian by discarding the non-diagonal terms. Neglecting the higher order terms in the expansion then (27) reduces to the form

$$\delta E(w, S) = \frac{1}{2} \sum_i H_{ii} \delta w_i^2. \quad (29)$$

If a parameter having an initial value $w_i$ is set to zero, then the increase in $E$ will be approximately given by the quantities $\frac{1}{2} H_{ii} \delta w_i^2$ that can be interpreted as a statistical significance measure (see Finoff, Hergert and Zimmermann 1993).

An implementation of this pruning procedure would characteristically consists of the following steps:

- **first**, select a relatively large initial network model [i.e. relatively large $J$],
- **second**, train the network model in the usual way [for example by backpropagation of gradient descent errors] until some stopping criterion is satisfied,
- **third**, compute the second derivatives $H_{ii}$ for each of the parameters, and thus evaluate the saliences $\frac{1}{2} H_{ii} \delta w_i^2$.
- **fourth**, sort the parameters by saliency and delete some of the low-saliency weights,
- **fifth**, go to the second step and repeat until some overall stopping criterion is satisfied.

Clearly, there are various choices to be made. The most important consideration, however, is to decide upon an appropriate number of parameters to be removed. The choice can be influenced by the number of pruning steps already performed as well as by visual inspection of the distribution of the test measure (see Finoff, Hergert and Zimmermann 1993). If this problem is solved satisfactorily the pruning technique that is generally performed interactively reduces overfitting and improves generalization of neural spatial interaction models.

The second approach for directing network architecture modification and selection is through the use of regularization which involves the addition of an extra term $R(w)$ to the error function $E(w)$ which is designed to penalize mappings that are not smooth. With a sum-of-squares error function, the total error function, $E(w, S)$, to be minimized becomes
The parameter $\mu [\mu \in [0, \infty)]$ controls the degree of regularization, i.e. the extent to which the penalty term $R(w)$ influences the form of the solution. Training is performed by minimizing the total error function $\tilde{E}(w, S)$ that requires the derivatives of $R(w)$ with respect to the model parameters to be computed efficiently. A spatial interaction function $\Omega$ that provides a good fit to the training data will give a small value for $E(w, S)$, while one that is very smooth will give a small value for $R(w)$. The resulting network model is a compromise between fitting the data and minimizing $R(w)$. One of the simplest regularizers $R(w)$ is called weight decay and consists of the sum of squares of the adaptive model parameters

$$R(w) = \frac{1}{2} \sum_i w_i^2$$

(31)

the first derivative of which leads to the weight decay in the weight updates (see Hanson and Pratt 1989). The use of this form of regularizer corresponds to ridge regression in conventional curve fitting. Hinton (1987) has empirically shown that such a regularizer can lead to significant improvements in network generalization.

One of the difficulties of the simple regularizer (31) is that it tends to favour many small parameter values rather than a few large ones. This problem can be overcome by using a modified penalty term of the form

$$R(w) = \sum_i \frac{\hat{w}_i^2}{\hat{w}_i^2 + \hat{w}_i^2}$$

(32)

where $\hat{w}$ corresponds to a parameter that sets a scale usually chosen by hand to be of order unity. Use of this penalty term has been called weight elimination (Weigend, Huberman and Rumelhart 1990). It will tend to favour a few large parameter values rather than many small ones, and, thus, is more likely to eliminate parameters from the model than (32). This leads to a form of network model pruning which is combined with the training process itself rather than alternating with it as in the case of pruning techniques.

A principal alternative to regularization and weight pruning as a way of controlling the effective complexity of a neural spatial interaction model is the procedure of stopped or cross-validation training. This method, in which an oversized network model is trained until the error on a further validation data set deteriorates, then training is stopped, is a true innovation coming out of neural network research since model selection does not require convergence of the training process. The training process is used here to perform a directed search of the parameter space for a model that does not overfit the data and, thus,
demonstrates superior generalization performance. Various theoretical and empirical results have provided strong evidence for efficiency of cross-validation training (Weigend, Rumelhart and Huberman 1991, Baldi and Chauvin 1991, Finnoff 1991, Fischer and Gopal 1994a, b). Although many questions remain, a picture is starting to emerge as to the mechanisms that are responsible for the effectiveness of this procedure. In particular, it has been shown that stopped training has the same sort of regularization effect (i.e. reducing model variance at the cost of bias) that penalty terms provide.

Since the efficiency of a model selection technique may depend on the data fitting problem at hand, a truly informative performance comparison can only be obtained by testing on a wide range of examples containing varying degrees of complexity.

6 Evaluating the Generalisation Performance of a Neural Spatial Interaction Model

The third stage in the model building process corresponds to testing the out-of-sample [generalisation] performance of the model. First, we consider the standard approach, and then evaluation by means of bootstrapping.

The Standard Approach

The standard approach for finding a good neural spatial interaction model [see Fischer and Gopal 1994a] is to split the available set of samples into three sets: training, validation, and test sets. The training set is used for parameter estimation. In order to avoid overfitting, a common procedure is to use a network model with sufficiently large $J$ for the task, to monitor – during training – the out-of-sample performance on a separate validation set, and finally to choose the network model that corresponds to the minimum on the validation set, and employ it for future purposes such as the evaluation on the test set. It has been common practice in the neural network community to fix these sets. But recent experience has found this approach to be very sensitive to the specific splitting of the data. Thus, usual tests of out-of-sample or generalization reliability may appear over optimistic.

Randomness enters in two ways in neural spatial interaction modelling: in the splitting of the data samples on the one side and in choices about the parameter initialization and the control parameters of the estimation approach utilized [such as $x$ in the global search procedure described in Section 4] on the other. This leaves one question widely open. What is the variation in out-of-sample performance as one varies training, validation and test sets? This is an important question since real world problems do not come with a tag on each pattern telling how it should be used. Thus, it is useful to vary both the data partitions and
parameter initializations to find out more about the distributions of out-of-sample errors.

Monte Carlo experiments can provide certain limited information on the behaviour of the test statistics. The limitation of Monte Carlo experiments is that any results obtained pertain only to the environment in which the experiments are carried out. In particular, the data-generating mechanism has to be specified, and it is often difficult to know whether any given data-generating mechanism is to any degree representative for the empirical setting under study.

**Evaluation by Means of Bootstrapping**

Motivated by the desire to obtain distributional results for the test statistics that rely neither on large size approximations nor on artificial data generating assumptions, statisticians have developed resampling techniques such as bootstrapping that permit rather accurate estimation of finite sample distributions for test statistics of interest. Bootstrapping is a computer intensive non-parametric approach to statistical inference that enables to estimate standard errors by resampling the data in a suitable way (see Efron and Tibshirani 1993). This idea can be applied to the third stage in neural spatial interaction modelling in two different ways. One can consider each input-output pattern as a sampling unit, and sample with replacement from the input-output pairs to create a bootstrap sample. This is sometimes called bootstrapping pairs (Efron and Tibshirani 1993) since the input-output pairs remain intact, and are resampled as full patterns. On the other hand, one can consider the predictors as fixed, treat the model residuals \( y_k - \Omega(x_k, \hat{\mathbf{w}}) \) as the sampling units, and create a bootstrap sample by adding residuals to the model fit \( \Omega(x_k, \hat{\mathbf{w}}) \). This is termed the bootstrap residual approach. In this approach, the residuals obtained from one specific model are used in rebuilding patterns to obtain error bars reflecting all sources of error, including model misspecification. If interest is discovering variation due to data samples rather than error bars, then the bootstrapping pairs approach is more appropriate.

The bootstrapping approach may be summarized as outlined in Fig. 2. The idea behind this approach is to generate many pseudo-replicates of the training, validation and test sets, then re-estimating the model parameters \( \mathbf{w} \) on each training bootstrap sample, and testing the out-of-sample performance on the test bootstrap samples. In this bootstrap world, the errors of forecast, and the errors in the parameter estimates, are directly observable. The Monte-Carlo distribution of such errors can be used to approximate the distribution of the unobservable errors in the real parameter estimates and the real forecasts. This approximation is the bootstrap: it gives a measure of the statistical uncertainty in the parameter estimates and the forecasts. Focus in the context of the third stage of model building is laid on the performance of the forecasts measured in terms of \( ARV \).
A diagram of the bootstrap procedure for estimating the standard error of the generalization performance of neural spatial interaction models.

Original Data Set

$S = \{ (x_1, y_1), \ldots, (x_K, y_K) \}$

Generate Bootstrap Samples

$S_{\text{Train}}^{*1}, S_{\text{Valid}}^{*1}, S_{\text{Test}}^{*1}$

$S_{\text{Train}}^{*B}, S_{\text{Valid}}^{*B}, S_{\text{Test}}^{*B}$

Minimize for each training bootstrap sample

$\sum_{k=1, \ldots, K} \left[ y_{k}^{*b} - \Omega(x_{k}^{*b}, w) \right]^2$

Bootstrap Replications

$\text{ARV}^*(S_{\text{Train}}^{*1})$  \ldots  $\text{ARV}^*(S_{\text{Test}}^{*B})$

Bootstrap Estimate of Standard Error

$\widehat{\text{se}}_B = \left\{ \sum_{b=1, \ldots, B} \left[ \text{ARV}^*(S_{\text{Train}}^{*b}) - \sum_{d=1, \ldots, B} \text{ARV}^*(S_{\text{Train}}^{*d}) \right]^2 / (B-1) \right\}^{1/2}$

Fig. 2  A diagram of the bootstrap procedure for estimating the standard error of the generalization performance of neural spatial interaction models.
In more detail the approach may be described by the following steps: The details may be a bit complicated, but the main idea is straightforward:

**Step 1:** Conduct three totally independent resampling operations in which

(i) \( B \) independent *training bootstrap samples* are generated, by randomly sampling \( K_1 \) times, with replacement, from the observed input-output pairs \( S = \{(x_i, y_i), (x_j, y_j), \ldots, (x_K, y_K)\}: \)

\[
S_{\text{train}}^{b} = \{(x_{i_1}^b, y_{i_1}^b), (x_{i_2}^b, y_{i_2}^b), \ldots, (x_{i_{K_1}}^b, y_{i_{K_1}}^b)\}
\] (33)

for \( k_1, k_2, \ldots, k_{K_1} \) a random sample of integers 1 through \( K_1 < K \) and \( b = 1, \ldots, B \),

(ii) \( B \) independent *validation bootstrap samples* are formed, by randomly sampling \( K_2 \) times, with replacement, from the observed input-output pairs \( S: \)

\[
S_{\text{valid}}^{b} = \{(x_{i_1}^b, y_{i_1}^b), (x_{i_2}^b, y_{i_2}^b), \ldots, (x_{i_{K_2}}^b, y_{i_{K_2}}^b)\}
\] (34)

for \( k_1, k_2, \ldots, k_{K_2} \) a random sample of integers 1 through \( K_2 < K_1 < K \) and \( b = 1, \ldots, B \),

(iii) \( B \) independent *test bootstrap samples* are formed, by randomly sampling \( K_3 \) times, with replacement, from the observed input-output pairs \( S: \)

\[
S_{\text{test}}^{b} = \{(x_{i_1}^b, y_{i_1}^b), (x_{i_2}^b, y_{i_2}^b), \ldots, (x_{i_{K_3}}^b, y_{i_{K_3}}^b)\}
\] (35)

for \( k_1, k_2, \ldots, k_{K_3} \) a random sample of integers 1 through \( K_3 = K_2 < K_1 < K \) and \( b = 1, \ldots, B \),

**Step 2:** For each training bootstrap sample \( S_{\text{train}}^{b} \) \((b = 1, \ldots, B)\) minimize

\[
\sum_{i=1}^{K_1} \left| y_{i}^{b} - \Omega_{\text{train}}(x_{i}^{b}, w) \right|^2
\] (36)

with an appropriate parameter estimation procedure. During the training process the ARV performance of the model is monitored on the bootstrap validation set. The training process is stopped when
ARV* (S_{test}^b) starts to increase. This yields bootstrap parameter estimates \( \hat{w}^* \).

**Step 3:** Calculate the bootstrap ARV-Statistic of generalization performance, \( \hat{ARV}^* (S_{test}^b) \), for each test bootstrap sample. The distribution of the pseudo-errors \( \hat{ARV}^* - ARV \) can be computed, and used to approximate the distribution of the real errors \( \hat{ARV}^* - ARV \). This approximation is the bootstrap.

**Step 4:** The variability of \( \hat{ARV}^* (S_{test}^b) \) for \( b=1,...,B \) gives an estimate of the expected accuracy of the model performance. Thus, estimate the standard error of the generalization performance statistic by the sample standard deviation of the \( B \) bootstrap replications:

\[
\hat{se}_a = \left\{ \frac{1}{B} \sum_{b=1}^{B} [\hat{ARV}^* (S_{test}^b) - \hat{ARV}^* (.)]^2 / (B - 1) \right\}^{1/2}
\]

(37) where

\[
\hat{ARV}^* (.) = \sum_{b=1}^{B} \hat{ARV}^* (S_{test}^b).
\]

(38)

Note that the name 'bootstrap' refers to the use of the original sample pairs to generate new data sets. The procedure requires to retrain the neural spatial interaction model \( B \) times (see Step 2). Typically, the \( B \) is in the range \( 20 \leq B \leq 200 \) (see Tibshirani 1996). Increasing \( B \) further does not bring a substantial reduction in variance (Efron and Tibshirani 1993).

7 Concluding Remarks

The exuberance of interest in neural network research has been accompanied by no little measure of hyperbole concerning the technological potential of computational neural networks. In addition, a definite mystique perceived by those outside the neural network community arises from the origins of computational neural networks in the study of natural neural systems, and in the associated metaphorical jargon of the field. Unfortunately, both the hyperbole and the mystique have evidently acted to lessen the amount of serious attention given to computational neural network modelling in spatial analysis. Much of the
research that has been done so far in these fields has been carried out by only a handful of researchers.

Neural spatial interaction models are attractive non-linear models. Despite all the progress made in the past few there are several areas that need further research. Finding a suitable neural spatial interaction model for a particular application, for example, is still a challenging task. It is possible to experiment with hand crafted designs found by trial and error procedures. But a much more appealing approach would be to employ techniques that tune the network topology automatically. Suggestions for work along these lines have already begun appearing. Fischer and Leung (1998), for example, suggest to combine neural network topology optimisation performed by genetic algorithms with gradient descent backpropagation for parameter estimation. The potential of using evolutionary programming rather than genetic algorithms for model selection is an open area of research.

Up to now research activities had been focused exclusively on unconstrained neural spatial interaction modelling. In many practical situations we have, in addition to \((x_k, y_k)\)-data samples, information about the inflow and/or outflow totals which the mapping should satisfy. This is referred to as prior knowledge, and its inclusion in the network model design process can often lead to substantial improvements in performance. In principle, prior knowledge can be incorporated into the network structure itself or into the pre-processing and post-processing stages. It can also be used to modify the training process through the use of regularization. Further research activities are necessary to identify a generic way of handling constraints in neural spatial interaction modelling.

**Acknowledgement:** The author gratefully acknowledges the grant no. P12681-INF provided by the Fonds zur Förderung der wissenschaftlichen Forschung (FWF).
References


