PRINCIPLES OF NEURAL SPATIAL INTERACTION MODELLING

Manfred M. Fischer
Vienna University of Economics and Business
manfred.fischer@wu.ac.at

Keywords: Spatial interaction, neural networks, non-linear function approximation, model performance, network learning
JEL Classification: C31, C45, R19

ABSTRACT:

The focus of this paper is on the neural network approach to modelling origin-destination flows across geographic space. The novelty about neural spatial interaction models lies in their ability to model non-linear processes between spatial flows and their determinants, with few – if any – a priori assumptions of the data generating process. The paper draws attention to models based on the theory of feedforward networks with a single hidden layer, and discusses some important issues that are central for successful application development. The scope is limited to feedforward neural spatial interaction models that have gained increasing attention in recent years. It is argued that failures in applications can usually be attributed to inadequate learning and/or inadequate complexity of the network model. Parameter estimation and a suitably chosen number of hidden units are, thus, of crucial importance for the success of real world applications. The paper views network learning as an optimization problem, describes various learning procedures, provides insights into current best practice to optimize complexity and suggests the use of the bootstrap pairs approach to evaluate the model's generalization performance.

1. INTRODUCTION

The development of spatial interaction models is one of the major intellectual achievements and, at the same time, perhaps the most useful contribution of spatial analysis to social science literature. Since the pioneering work of Wilson (1970) on entropy maximization, there have been surprisingly few innovations in the design of spatial interaction models. Fotheringham's (1983) competing destinations version, Griffith's eigenvector spatial filter versions[1] (see Griffith 2003; Fischer and Griffith 2008), the spatial econometric interaction models[2] (see LeSage and Pace 2009; LeSage and Fischer 2010), and neural network based (briefly neural) spatial interaction models (see Fischer and Gopal 1994; Fischer 2002) are the principal exceptions.

The focus in this paper is on neural networks as efficient non-linear tools for modelling interactions across geographic space. The term "neural network" has its origins in attempts to find mathematical representations of information processing in the study of natural neural systems (McCulloch and Pitts 1943; Rosenblatt 1962). Indeed, the term has been used very broadly to include a wide range of different model structures, many of which have been the subject of exaggerated claims to mimic neurobiological reality[3]. As rich as neural networks are, they still ignore a host of biologically relevant features. From the perspective of applications in spatial interaction modelling, however, neurobiological realism is not necessary. In contrast, it would impose entirely unnecessary constraints.

From the statistician's point of view neural network models are analogous to non-parametric, non-linear regression models. The novelty about neural spatial interaction models lies in their ability to model non-linear processes with few – if any – *a priori* assumptions about the nature of the data generating process. We limit ourselves to models known as feedforward neural models[4]. Spatial interaction models of this kind can be viewed as a general framework for non-linear function approximation where the form of the mapping is governed by a number of adjustable parameters. The network inputs are origin, destination and separation variables, and the network weights the model parameters.

[1] Eigenvector spatial filtering (see Griffith 2003) enables spatial autocorrelation effects to be captured, and shifts attention to spatial autocorrelation arising from missing origin and destination factors reflected in flows between pairs of locations.

[2] Note that spatial econometric interaction models are – in general – formally equivalent to regression models with spatially autocorrelated errors, but differ in terms of the data analysed and the manner in which the spatial weights matrix is defined.

[3] Neural networks can model cortical local learning and signal processing, but they are not the brain, neither are many special purpose systems to which they contribute (Weng and Hwang 2006).

[4] Feedforward neural networks are sometimes also called multilayer perceptrons even though the term perceptron is usually used to refer to a network with linear threshold gates rather than with continuous non-linearities. Radial basis function networks, recurrent networks rooted in statistical physics, self-organizing systems and ART [Adaptive Resonance Theory] models are other important classes of neural networks. For a fuzzy ARTMAP multispectral classifier see, for example, Gopal and Fischer (1997).

The paper is organized as follows. The next section continues to provide the context in which neural spatial interaction modelling is considered. Neural spatial interaction models that have a single hidden layer architecture with K input nodes (typically, K=3) and a single output node are described in some detail in Section 3. They represent a rich and flexible class of universal approximators. Section 4 proceeds to view the problem of determining the network parameters within a framework that involves the solution of a non-linear optimization problem with an objective function that recognizes the integer nature of the origin-destination flows. The section that follows reviews some of the most important training (learning) procedures and modes that utilize gradient information for solving the problem. This requires the evaluation of derivatives of the objective function – known as error or loss function in the machine-learning literature[5] – with respect to the network parameters.

Section 6 addresses the issue of network complexity and briefly discusses some techniques to determine the number of hidden units. This problem is shown to essentially consist of optimizing the complexity of the neural spatial interaction model (complexity in terms of free parameters) in order to achieve the best generalization performance. Section 7 then moves attention to the issue of how to appropriately test the generalization performance of the estimated neural spatial interaction model. Some conclusions and an outlook for the future are given in the final section.

2. CONTEXT

Spatial interaction models of the gravity type represent a class of models used to explain origin-destination flows across geographic space. Examples include migration, journey-to-work and shopping flows, trade and commodity flows, information and knowledge flows. Origin and destination locations of interaction represent points or areas (regions) in geographic space. Such models typically recognize three types of factors to explain mean interaction frequencies between origin and destination locations: (i) origin-destination variables that characterize the way spatial separation of origins from destinations constrains or impedes the interaction, (ii) origin-specific variables that characterize the ability of the origins to produce or generate flows, and (iii) destination-specific variables that represent the attractiveness of destinations.

Suppose we have a spatial system consisting of n regions, where i denotes the origin region $(i=1, ..., n)$ and j the destination region $(j=1, ..., n)$. Let $m(i, j)$ $(i, j=1, ..., n)$ denote observations on random variables, say $M(i, j)$, each of which corresponds to flows of people, commodities, capital, information or knowledge from region i to region j. The $M(i, j)$ are assumed to be independent random variables. They are sampled from a specified probability distribution that is dependent upon some mean, say $\mu(i, j)$. Let us assume that no a priori information is given about the row and column totals of the observed flow matrix $[m(i, j)]$. Then the mean interaction frequencies between origin i and destination j may be modelled by

$$\mu(i, j) = C\ A(i)^{\alpha}\ B(j)^{\beta}\ F(i, j) \qquad i, j=1, ..., n \quad (1)$$

where $\mu(i, j) = E[M(i, j)]$ is the expected flow, C denotes a constant term, the quantities $A(i)$ and $B(j)$ are called origin and destination variables, respectively. α and β indicate their relative importance, and $F(i, j)$ represents a distance deterrence function that constitutes the very core of spatial interaction models. Hence, a number of alternative specifications of $F(\cdot)$ have been proposed in the literature (see, for example, Sen and Smith 1995, pp. 92-99). But the negative exponential function is the most popular choice (with theoretical relevance from a behavioural viewpoint):

$$F(i, j) = \exp[-\theta\ d(i, j)] \qquad i, j=1, ..., n \quad (2)$$

where θ denotes the so-called distance sensitivity parameter that has to be estimated.

Inserting Eq. (2) into Eq. (1) yields the well known class of exponential spatial interaction models that can be expressed equivalently as a log-additive model of the form

$$Y(i, j) = \kappa + \alpha\ a(i) + \beta\ b(j) + \theta\ d(i, j) + \varepsilon(i, j) \quad (3)$$

where $Y(i, j) \equiv \log[\mu(i, j)]$, $\kappa \equiv \log C$, $a(i) \equiv \log[A(i)]$ and $b(j) \equiv \log[B(j)]$. Of note is that the back transformation of this log-linear specification results in an error structure of the exponential spatial interaction model being multiplicative. The parameters κ, α, β and θ have to be estimated if future flows are to be predicted.

There are n^2 equations of the form (3). Using matrix notation we may write these equations more compactly as

$$Y = X\ \theta + \varepsilon \quad (4)$$

where Y denotes the N-by-1 vector of observations on the interaction variable, with $N = n^2$ (see Table 1 for the data organization convention). X is the N-by-4 matrix of observations on the explanatory variables including the origin, destination, separation variables, and the intercept. θ is the associated 4-by-1 parameter vector, and the N-by-1 vector $\varepsilon = [\varepsilon(1,1), ..., \varepsilon[n,n)]^T$ denotes the vectorized form of $[\varepsilon(i, j)]$.

If the spatial interaction model given by Eq. (4) is correctly specified, then provided that the regressor variables are not perfectly collinear, θ is estimable under the assumption that the error terms are *iid* with zero mean and constant variance, and the OLS estimator is the best linear unbiased estimator. A violation of these assumptions may lead to spatial autocorrelation.

[5] We will use the terms error function, loss function and cost function interchangeably in this paper.

It is noteworthy that the above spatial interaction model can not guarantee that the predicted flows when summed by rows or columns of the spatial interaction data matrix will necessarily have the property to match observed totals leaving the origins *i* $(i = 1, ..., n)$ or terminating at the destinations j $(j = 1, ..., n)$ in the given spatial interaction system. If the outflow totals for each origin zone and/or the inflow totals into each destination zone are a priori known, then the log-linear model given by Eq. (4) would need to be modified to incorporate the explicitly required constraints to match exact totals. Imposing origin and/or destination constraints leads to so-called production-constrained, attraction-constrained and production-attraction-constrained spatial interaction models that may be convincingly justified using entropy maximizing methods (see Wilson 1967).

Dyad Label	ID$_{\text{origin}}$	ID$_{\text{destination}}$	Flow	Origin Variable	Destination Variable	Separation (Origin, Destination)
1	1	1	$Y(1, 1)$	$a(1)$	$b(1)$	$d(1, 1)$
2	2	1	$Y(2, 1)$	$a(2)$	$b(1)$	$d(2, 1)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	n	1	$Y(n, 1)$	$a(n)$	$b(1)$	$d(n, 1)$
n+1	1	2	$Y(1, 2)$	$a(1)$	$b(2)$	$d(1, 2)$
n+2	2	2	$Y(2, 2)$	$a(2)$	$b(2)$	$d(2, 2)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2n	n	2	$Y(n, 2)$	$a(n)$	$b(2)$	$d(n, 2)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
(n-1)n	1	n	$Y(1, n)$	$a(1)$	$b(n)$	$d(1, n)$
(n-1)n+1	2	n	$Y(2, n)$	$a(2)$	$b(n)$	$d(2, n)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
n^2	n	n	$Y(n, n)$	$a(n)$	$b(n)$	$d(n, n)$

Table 1. Data organization convention

Moreover, note that this widely used log-normal specification of the spatial interaction model has several shortcomings[6]. Most importantly, it suffers 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.

3. FEEDFORWARD NEURAL SPATIAL INTERACTION MODELS

Neural spatial interaction models represent the most recent innovation in the design of spatial interaction models. For concreteness and simplicity, we consider neural spatial interaction models based on the theory of single hidden layer feedforward networks. Single hidden layer feedforward neural networks consist of nodes (also known as processing units or simply units) that are organized in layers. Figure 1 shows a schematic diagram of a typical feedforward neural spatial interaction model containing a single intermediate layer of processing units separating input from output units. Intermediate layers of this sort are called *hidden* layers to distinguish them from the input and output layers. In this network there are three input nodes representing the origin, destination and separation variables (denoted by x_1, x_2, x_3); H hidden units (say $z_1, ..., z_H$) representing hidden summation units (denoted by the symbol $\textcircled{\Sigma}$); and one (summation) output node representing origin-destination flows. Weight parameters are represented by links between the nodes. Observe the feedforward structure where the inputs are connected only to units in the hidden layer, and the outputs of this layer are connected only to the output layer that consists of only one unit.

Any network diagram can be converted into its corresponding mapping function, provided that the diagram is feedforward as in Fig. 1 so that it does not contain closed directed cycles[7]. This guarantees that the network output can be described by a series of functional transformations as follows. First, we form a linear combination[8] of the K input variables $x_1, ..., x_K$ (typically K=3) to get the input, say net_h, that hidden unit *h* receives

$$net_h = \sum_{k=1}^{K} w_{hk}^{(1)} x_k + w_{ho}^{(1)} \qquad (5)$$

for $h = 1,...,H$. The superscript $^{(1)}$ indicates that the corresponding parameters are in the first parameter layer of the network. The parameters $w_{hk}^{(1)}$ represent connection weights going from input k $(k = 1,...,K)$ to hidden unit h $(h = 1,...,H)$, and $w_{ho}^{(1)}$ is a bias[9].

These quantities, net_h, are known as activations in the field of neural networks. Each of them is then transformed using a non-linear *transfer* or *activation* function[10] φ to give the output

[6] Flowerdew and Aitkin (1982), for example, question the appropriateness of this model specification, and suggest instead that the observed flows follow a Poisson distribution, leading to models termed Poisson spatial interaction models.

[7] Networks with closed directed cycles are called *recurrent* networks. There are three types of such networks: *first*, networks in which the input layer is fed back into the input layer itself; *second*, networks in which the hidden layer is fed back into the input layer, and *third*, networks in which the output layer is fed back into the input layer. These feedback networks are useful when input variables represent time series.

[8] Note, we could alternatively use product rather than summation hidden units to supplement the inputs to a neural network with higher-order combinations of the inputs to increase the capacity of the network in an information capacity sense. These networks are called *product unit* rather than summation unit networks (see Fischer and Reismann 2002b).

[9] This term should not be confused with the term bias in a statistical sense.

[10] The inverse of this function is called link function in the statistical literature. Note that radial basis function networks may be viewed as single hidden layer networks that use radial basis function nodes in the hidden layer. This class of

$$z_h = \varphi(net_h) \tag{6}$$

for $h = 1,...,H$. These quantities are again linearly combined to generate the input, called net, that the output unit receives

$$net = \sum_{h=1}^{H} w_h^{(2)} z_h + w_o^{(2)} . \tag{7}$$

The superscript $^{(2)}$ indicates that the corresponding parameters are in the second parameter layer of the network. The parameters $w_h^{(2)}$ represent the connection weights from hidden units h $(h = 1,...,H)$ to the output unit, and $w_o^{(2)}$ is a bias parameter. Finally, net is transformed to produce the output $\psi(net)$, where ψ denotes an activation function of the output unit.

Information processing in such networks is, thus, straightforward. The input units just provide a 'fan-out' and distribute the input to the hidden units. These units sum their inputs, add a constant (the bias) and take a fixed transfer function φ_h of the result. The output unit is of the same form, but with output activation function ψ. Network output can then be expressed in terms of an output function

$$\phi_H(x,w) = \psi\left[\sum_{h=1}^{H} w_h^{(2)} \varphi_h\left(\sum_{k=1}^{K} w_{hk}^{(1)} x_k + w_{k0}^{(1)}\right) + w_0^{(2)}\right] \tag{8}$$

where the expression $\phi_H(x,w)$ is a convenient short-hand notation for the model output since this depends only on inputs and weights. Vector $x = (x_1, ..., x_K)$ is the input vector and w represents the vector of all the weights and bias terms. $\varphi(\cdot)$ is a non-linear [generally sigmoid] hidden layer activation function and $\psi(\cdot)$ an output unit [often quasi-linear] activation function, both continuously differentiable of order two on $\mathbb{R}$. The function ϕ is explicitly indexed by the number of hidden units, H, in order to indicate the dependence, but will be dropped for convenience.

Note that the bias terms $w_{k0}^{(1)}$ and $w_0^{(2)}$ in Eq. (8) can be absorbed[11] into the set of weight parameters by defining additional input and hidden unit variables, x_0 and z_0, whose values are clamped at one so that $x_0 = 1$ and $z_0 = 1$. Then the network function given by Eq. (8) becomes

$$\phi_H = \psi\left[\sum_{h=0}^{H} w_h^{(2)} \varphi_h\left(\sum_{k=0}^{K} w_{hk}^{(1)} x_k\right)\right] . \tag{9}$$

neural networks asks for a two stage approach for training. In the first stage the parameters of the basis functions are determined, while in the second stage the basis functions are kept fixed and the second layer weights are found (see Bishop 1995, 170 pp.).

[11] This is the same idea as incorporating the constant term in the design matrix of a regression by inserting a column of ones.

The main power of neural spatial interaction models accrues from their capability for universal function approximation. Cybenko (1989); Funahashi (1989); Hornik, Stinchcombe and White (1989) and many others have shown that single hidden layer neural networks such as those given by Eq. (9) can approximate arbitrarily well any continuous function.

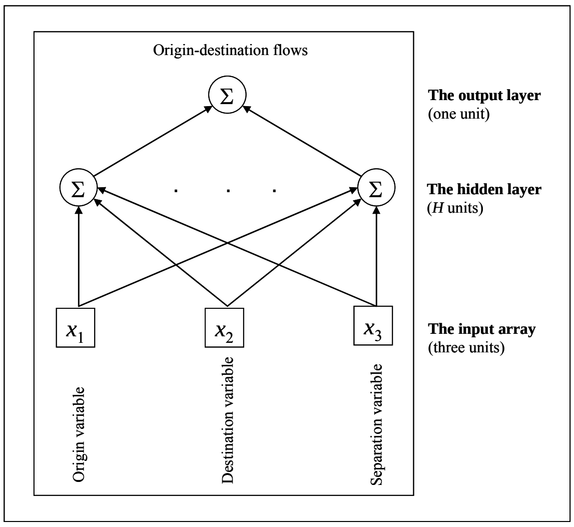

Figure 1. Network diagram of the neural spatial interaction model as defined by Eq. (8), for *K*=3 (bias units deleted)

Neural spatial interaction modelling involves three major stages (Fischer and Gopal 1994):

- The *first stage* consists of the identification of a model candidate from the general class of neural spatial interaction models of type (9). This involves both the specification of appropriate transfer functions ψ and φ, and the number, *H*, of hidden units.
- The *second stage* involves solving the network training [network learning, parameter estimation] problem, and hence determines the optimal set of model parameters where optimality is defined in terms of an error [loss, performance] function.
- The *third stage* is concerned with testing and evaluating the out-of-sample [generalization] performance of the chosen model.

Both the theoretical and practical side of the model selection problem have been intensively studied (see Fischer 2001, 2000 among others). The standard approach for finding a good neural spatial interaction model is to split the available set of samples into three subsets: 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 *H* for the task at hand, to monitor – during training – the out-of-sample performance on a separate validation set, and finally to choose the model that corresponds to the minimum on the validation set, and employ it for future purposes such as the evaluation on the test set.

4. A RATIONALE FOR THE ESTIMATION APPROACH

If we view a neural spatial interaction model as generating a family of approximations (as *w* ranges over *W*, say) to an unknown spatial interaction function, then we need a way to pick a best approximation from this family. This is the function of network learning or network training which might be viewed as an optimization problem[12].

We develop a rationale for an appropriate objective (loss or 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 *D*. We can express this as attempting to maximize the term

$$P(\phi(w) \mid D) = \frac{P(D \mid \phi(w))\; P(\phi(w))}{P(D)} \tag{10}$$

where ϕ represents the neural spatial interaction model (with w denoting the vector of weights) in question. $P(D \mid \phi(w))$ is the probability that the model would have produced the observed data *D*. Since sums are easier to work with than products, we will maximize the log of this probability, and since the log is a strictly monotonic transformation, maximizing the log is equivalent to maximizing the probability itself. In this case we have

$$\begin{aligned}&\log P(\phi(w) \mid D) = \\ &\log P(D \mid \phi(w)) + \log P(\phi(w)) - \log P(D).\end{aligned} \tag{11}$$

The probability of the data, $P(D)$, is not dependent on the model. Thus, it is sufficient to maximize $\log P(D \mid \phi(w)) + \log P(\phi(w))$. The first of these terms represents the probability of the data given the model, and hence measures how well the neural network model accounts for the data. The second term is a representation of the model itself; that is, it is *a priori* probability, that can be utilized 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, $D = \{q_u = (x_u, y_u) : u = 1, ..., U\}$, each q_u we will assume to be chosen independently of the others. Hence we can write the probability of the data given the model as

$$\begin{aligned}&\log P(D \mid \phi(w)) = \\ &\log \prod_u P(q_u \mid \phi(w)) = \sum_u \log P(q_u \mid \phi(w)).\end{aligned} \tag{12}$$

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

$$\begin{aligned}&\log P(D|\phi(w)) = \\ &\sum_u \log P\left(y_u \middle| x_u \text{ and } \phi(w)_u\right) + \sum_u \log P(x_u).\end{aligned} \tag{13}$$

Since we assume that x_u does not depend on the model, the second term of the right hand side of the equation will not affect the determination of the optimal model. Thus, we need only to maximize the term $\sum_u \log P(y_u \mid x_u \text{ and } \phi(w)_u)$.

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_u and the observed target data y_u, a probabilistic assumption. We assume that the relationship between x_u and y_u is not deterministic, but that for any given x_u there is a distribution of possible values of y_u. But the model is deterministic, so rather than attempting to predict the actual outcome we only attempt to predict the expected value of y_u given x_u. Therefore, the model output is to be interpreted as the mean bilateral interaction frequencies (that is, those from the location of origin to the location 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 is the realization of a sequence of independent Poisson random variables. Under this assumption we can write the probability of the data given the model as

$$P\left(y_u \middle| x_u \text{ and } \phi(w)_u\right) = \prod_u \frac{\phi(w)_u^{y_u} \exp(-\phi(w)_u)}{y_u!} \tag{14}$$

and, hence, define a maximum likelihood estimator as a parameter vector $\hat{w}$ which maximizes the log-likelihood *L*

$$\begin{aligned}&\max_{w \in W} \; L(x, y, w) = \\ &\max_{w \in W} \sum_u [y_u \log \phi(w)_u - \phi(w)_u - \log(y_u!)].\end{aligned} \tag{15}$$

Instead of maximizing the log-likelihood it is more convenient to view learning as solving the minimization problem

[12] This directs attention to the literature on numerical optimization theory, with particular reference to optimization techniques that use higher-order information such as conjugate gradient procedures and Newton's method. The methods use the gradient vector (first-order partial derivatives) and/or the Hessian matrix (second-order partial derivatives) of the loss function to perform optimization, but in different ways. A survey of first-order and second-order optimization techniques applied to network training can be found in Cichocki and Unbehauen (1993).

$$\min_{w \in W} \lambda(x,y,w) = \min_{w \in W} \left[-L(x,y,w)\right] \tag{16}$$

where the loss (cost) function λ is the negative log-likelihood L. λ is continuously differentiable on the Q-dimensional real parameter space ($Q = HK + H + 1$) which is a finite dimensional closed bounded domain and, thus, compact.

5. LEARNING MODES AND PROCEDURES

It can be shown that $\lambda(w)$ assumes its weight minimum under certain conditions, but characteristically there exist many minima in real world applications all of which satisfy

$$\nabla\lambda(w) = 0 \tag{17}$$

where $\nabla\lambda$ denotes the gradient of λ. The minimum for which the value of λ is smallest is termed the global minimum and other minima are called local minima.

There are many ways to solve the minimization problem (16). Closed-form optimization via the calculus of scalar fields rarely admits a direct solution. A relatively new set of interesting techniques that use optimality conditions from calculus are based on evolutionary computation (Goldberg 1989; Fogel 1995). But gradient procedures which use the first partial derivatives $\nabla\lambda(w)$, so-called first order strategies, are most widely used. Gradient search for solutions gleans its information about derivatives from a sequence of function values. The recursion scheme is based on the formula[13]

$$w(\tau+1) = w(\tau) + \eta(\tau)\, d(\tau) \tag{18}$$

where τ denotes the iteration step. Different procedures differ from each other with regard to the choice of step length $\eta(\tau)$ and search direction $d(\tau)$, the former being a scalar called *learning* rate and the latter a vector of unit length.

The simplest approach to using gradient information is to assume $\eta(\tau)$ being constant and to choose the parameter update in Eq. (18) to comprise a small step in the direction of the negative gradient so that

$$d(\tau) = -\nabla\lambda(w(\tau))\,. \tag{19}$$

After each such update, the gradient is re-evaluated for the new parameter vector $w(\tau+1)$. Note that the loss function is defined with respect to a training set of size $U1$, say D_{U1}, to be processed to evaluate $\nabla\lambda$. One complete presentation of the entire training set during the training process is called an *epoch*. The training process is maintained on an epoch-by-epoch basis until the connection weights and bias terms of the network stabilize and the average error over the entire training set converges to some minimum.

Gradient descent optimization may proceed in one of two ways: pattern mode and batch mode. In the *pattern mode* weight updating is performed after the presentation of each training example. Note that the loss functions based on maximum likelihood for a set of independent observations comprise a sum of terms, one for each data point. Thus

$$\lambda(w) = \sum_{u1 \in U1} \lambda_{u1}(w) \tag{20}$$

where λ_{u1} is called the *local error* (loss) while λ the *global error* (loss), and pattern based gradient descent makes an update to the parameter vector based on one training example at a time so that

$$w(\tau+1) = w(\tau) - \eta\nabla\lambda_{u1}(w(\tau)). \tag{21}$$

Rumelhart et al. (1986) have shown that pattern based gradient descent minimizes Eq. (16), if the learning parameter η is sufficiently small. The smaller η, the smaller will be the changes to the weights in the network from one iteration to the next and the smoother will be the trajectory in the parameter space. This improvement, however, is attained at the cost of a slower rate of training. If we make the learning rate parameter η too large so as to speed up the rate of training, the resulting large changes in the parameter weights assume such a form that the network may become unstable.

In the *batch mode* of learning, parameter updating is performed after the presentation of all the training examples that constitute an epoch. From an online operational point of view, the pattern mode of training is preferred over the batch mode, because it requires less local storage for each weight connection. Moreover, given that the training patterns are presented to the network in a random manner, the use of pattern-by pattern updating of parameters makes the search in parameter space stochastic in nature which in turn makes it less likely to be trapped in a local minimum. On the other hand, the use of batch mode of training provides a more accurate estimation of the gradient vector $\nabla\lambda$. Finally, the relative effectiveness of the two training modes depends on the problem to be solved (Haykin 1994, 152 pp.)

For batch optimization there are more efficient procedures, such as conjugate gradient and quasi-Newton methods, that are much more robust and much faster than gradient descent (Nocedal and Wright 1999). Unlike steepest gradient, these algorithms have the characteristic that the error function always decreases at each iteration unless the parameter vector has arrived at a local or global minimum. Conjugate gradient methods achieve this by incorporating an intricate relationship between the direction and gradient vectors. The initial direction vector $d(0)$ is set equal to the negative gradient vector at the initial step $\tau = 0$. Each successive direction vector is then computed

[13] When using an iterative optimization algorithm, some choice has to be made of when to stop the training process. There are various criteria that might be used. For example, learning may be stopped when the loss function or the relative change in the loss function falls below a prespecified value.

as a linear combination of the current gradient vector and the previous direction vector. Thus,

$$d(\tau+1) = -\nabla\lambda(w(\tau+1)) + \gamma(\tau)\, d(\tau) \qquad (22)$$

where $\gamma(\tau)$ is a time varying parameter. There are various rules for determining $\gamma(\tau)$ in terms of the gradient vectors at time τ and $\tau+1$, leading to the Fletcher-Reeves and Polak-Ribière variants of conjugate gradient algorithms (see Press et al. 1992). The computation of the learning rate parameter $\eta(\tau)$ in the update formula given by Eq. (18) involves a line search, the purpose of which is to find a particular value of η for which the loss function $\lambda(w(\tau)+\eta\, d(\tau))$ is minimized, given fixed values of $w(\tau)$ and $d(\tau)$.

The application of Newton's method to the training of neural networks is hindered by the requirement of having to calculate the Hessian matrix and its inverse, which can be computationally expensive for larger network models[14]. The problem is further complicated by the fact that the Hessian matrix has to be non-singular for its inverse to be computed. Quasi-Newton methods avoid this problem by building up an approximation to the inverse Hessian over a number of iteration steps. The most commonly variants are the Davidson-Fletcher-Powell and the Broyden-Fletcher-Goldfarb-Shanno procedures (see Press et al. 1992).

Quasi-Newton procedures are today the most efficient and sophisticated (batch) optimization algorithms. But they require the evaluation and storage in memory of a dense matrix at each iteration step τ. For larger problems (more than 1,000 weights) the storage of the approximate Hessian can be too demanding. In contrast, the conjugate gradient procedures require much less storage, but an exact determination of the learning rate $\eta(\tau)$ and the parameter $\gamma(\tau)$ in each iteration τ, and, thus, approximately twice as many gradient evaluations as the quasi-Newton methods.

When the surface modelled by the loss function in its parameter space is extremely rugged and has many local minima, then a local search from a random starting point tends to converge to a local minimum close to the initial point. In order to seek out good local minima, a good training procedure must thus include both a gradient based optimization algorithm and a technique like random start that enables sampling of the space of minima. Alternatively, stochastic global search procedures might be used. Examples of such procedures include Alopex (see Fischer, Reismann and Hlavácková-Schindler 2003), genetic algorithms (see Fischer and Leung 1998), and simulated annealing. These procedures guarantee convergence to a global solution with a higher probability, but at the expense of slower convergence.

Finally, it is worth noting that the technique of error backpropagation provides a computationally efficient technique to calculate the gradient vector of a loss function for a feedforward neural network with respect to the parameters. This technique – sometimes simply termed backprop – uses a local message passing scheme in which information is sent alternately forwards and backwards through the network. Its modern form stems from Rumelhart, Hinton and Williams (1986), illustrated for gradient descent optimization applied to the sum-of-squares error function. It is important to recognize, however, that error backpropagation can also be applied to our loss function and to a wide variety of optimization schemes for weight adjustment other than gradient descent, both in pattern or batch mode.

6. NETWORK COMPLEXITY

So far we have considered neural spatial interaction models of type (9) with *a priori* given numbers of input, hidden and output units. While the number of input and output units is basically problem dependent, the number *H* of hidden units is a free parameter that can be adjusted to provide the best testing performance on independent data, called testing set. But the testing error is not a simple function of *H* due to the presence of local minima in the loss function. The issue of finding a parsimonious model for a real world problem is critical for all models but particularly important for neural networks because the problem of overfitting is more likely to occur.

A neural spatial interaction model that is too simple (i.e. small *H*), 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 and have a large variance. In either case, the performance of the network model 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 (Bishop 1995, p. 332; Fischer 2000). There are some ways to control the complexity of a neural network model, complexity in terms of the number of hidden units or, more precisely, in terms of the independently adjusted parameters. Practice in neural spatial interaction modelling generally adopts a trial and error approach that trains a sequence of neural networks with an increasing number of hidden units and then selects that one which gives the best predictive performance on a testing set[15].

There are, however, other more principled ways to control the complexity of a neural network model in order to avoid overfitting[16]. One approach is that of *regularization*, which involves adding a regularization term $R(w)$ to the loss

[14] Note that computational time rises with the square of *Q*, the dimension of the parameter space.

[15] Note that limited data sets make the determination of *H* more difficult if there is not enough data available to hold out a sufficiently large independent test sample.

[16] A neural network is said to be overfitted to the data if it obtains an excellent fit to the training data, but gives a poor representation of the unknown function which the neural network is approximating.

function in order to control overfitting, so that the total error function to be minimized takes the form

$$\tilde{\lambda}_p(w) = \lambda_p(w) + \mu R(w) \tag{23}$$

where μ is a positive real number, the so-called regularization parameter, that controls the relative importance of the data dependent error $\lambda_p(w)$, and $R(w)$ the regularization term, sometimes also called complexity term. This term embodies the *a priori* knowledge about the solution, and therefore depends on the nature of the particular problem to be solved. Note that $\tilde{\lambda}_p(w)$ is called the *regularized error* or *loss function*.

One of the simplest forms of a regularizer is defined as the squared Euclidean norm of the parameter vector *w* in the network, as given by

$$R(w) = \|w\|^2. \tag{24}$$

This regularizer[17] is known as weight decay function that penalizes larger weights. Hinton (1987) has found empirically that a regularizer of this form can lead to significant improvements in network generalization.

Sometimes, a more general regularizer is used, for which the regularized error or loss takes the form

$$\lambda(w) + \mu\|w\|^m \tag{25}$$

where *m*=2 corresponds to the quadratic regularizer given by Eq. (24). The case *m*=1 is known as the 'lasso' in the statistics literature (Tibshirani 1996). The regularizer given by Eq. (25) has the property that – if μ is sufficiently large – some of the parameter weights are driven to zero in sequential learning algorithms, leading to a sparse model. As μ is increased, so an increasing number of parameters are driven to zero.

One of the limitations of this regularizer is inconsistency with certain scaling characteristics of network mappings. If one trains a network using original data and one network using data for which the input and/or target variables are linearly transformed, then consistency requires that the regularizer should be invariant to re-scaling of the weights and to shifts of the biases (Bishop 2006, p. 258). A regularized loss function that satisfies this property is given by

$$\lambda(w) + \mu_1\|w_{q1}\|^m + \mu_2\|w_{q2}\|^m \tag{26}$$

where μ_1, μ_2 and *m* are regularization parameters. w_{q1} denotes the set of the weights in the first parameter layer, that is $w_{11}^{(1)},...,w_{h1}^{(1)},...,w_{HK}^{(1)}$, and w_{q2} those in the second layer, that is $w_1^{(2)},...,w_h^{(2)},...,w_H^{(2)}$.

[17] In conventional curve fitting, the use of this regularizer is termed *ridge regression*.

The more sophisticated control of complexity that regularization offers over adjusting the number of hidden units by trial and error is evident. Regularization allows complex neural network models to be trained on data sets of limited size without severe overfitting, by limiting the effective network complexity. The problem of determining the appropriate number of hidden units is, thus, shifted to one of determining a suitable value for the regularization parameter(s) during the training process.

The principal alternative to regularization as a way to optimize the model complexity for a given training data set is the procedure of *early stopping*. As we have seen in the previous sections, training of a non-linear network model corresponds to an iterative reduction of the loss (error) function defined with respect to a given training data set. For many of the optimization procedures used for network training (such as conjugate gradient optimization) the error is a non-decreasing function of the iteration steps τ. But the error measured with respect to independent data, called *validation data set*, say D_{U2}, often shows a decrease first, followed by an increase as the network starts to overfit, as illustrated in Fischer and Gopal (1994). Thus, training can be stopped at the point of smallest error with respect to the validation data, in order to get a network that shows good generalization performance. But, if the validation set is mall, it will give a relatively noisy estimate of generalization performance, and it may be necessary to keep aside another data set, the test set D_{U3}, on which the performance of the network model is finally evaluated.

This approach of stopping training before a minimum of the training error has been reached is another way of eliminating the network complexity. It contrasts with regularization because the determination of the number of hidden units does not require convergence of the training process. The training process is used here to perform a directed search in the weight space for a neural network model that does not overfit the data and, thus, shows superior generalization performance. Various theoretical and empirical results have provided strong evidence for the efficiency of early stopping (see, for example, Weigend, Rumelhart and Huberman 1991; Baldi and Chauvin 1991; Finnoff 1991; Fischer and Gopal 1994). Although many questions remain, a picture is starting to emerge as to the mechanisms responsible for the effectiveness of this approach. 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.

7. GENERALIZATION PERFORMANCE

Model performance may be measured in terms of Kullback and Leibler's (1951) information criterion, *KLIC*, which is a natural performance criterion for the goodness-of-fit of ML estimated models

$$KLIC\left(D_{U3}\right)=$$

$$\sum_{u3\in U3}\frac{y_{u3}}{\sum\limits_{u3'\in U3}^{U3} y_{u3'}}\ln\left[\frac{y_{u3}\left[\sum\limits_{u3'\in U3} y_{u3'}\right]^{-1}}{\phi\left(x_{u3},\hat{w}\right)\left[\sum\limits_{u3'\in U3}\phi\left(x_{u3'},\hat{w}\right)\right]^{-1}}\right] \qquad (27)$$

where $\left(x_{u3}, y_{u3}\right)$ denotes the $u3$-th pattern of the data set D_{U3}, and ϕ is the estimated neural spatial interaction model under consideration.

The standard approach to evaluate the out-of-sample (generalization or prediction) performance of a neural spatial interaction model (see Fischer and Gopal 1994) is to split the data set *D* of size *U* into three subsets: the *training* [in-sample] *set* $D_{U1}=\{q_{u1}=(x_{u1},y_{u1})\}$ of size $U1$, the internal validation set $D_{U2}=\{q_{u2}=(x_{u2},y_{u2})\}$ of size $U2$ and the testing [out-of-sample, generalization, prediction] set $D_{U3}=\{q_{u3}=(x_{u3},y_{u3})\}$ of size $U3$, with U1+U2+U3=U. The training set serves for parameter estimation. The validation set is used to determine the stopping point before overfitting occurs, and the test set to evaluate the generalization performance of the model, using some measure of error between a prediction and an observed value, such as $KLIC\left(D_{U3}\right)$.

Randomness enters into this standard approach to neural network modelling in two ways: in the splitting of the data samples, and in choices about the parameter initialization. This leaves one question wide open. What is the variation of test performance as one varies training, validation and test sets? This is an important question, since there is not just one 'best' split of the data or obvious choice for the initial weights. Thus, it is useful to vary both the data partitions and parameter initializations to find out more about the distribution of generalization errors. One way is to use the bootstrap pairs approach (Efron 1982) with replacement to evaluate the performance, reliability, and robustness of the neural spatial interaction model.

The bootstrap pairs approach[18] is an intuitive way to apply the bootstrap notion to combine the purity of splitting the data set into three data sets with the power of a resampling procedure. The basic idea of this approach is to generate *B* pseudo-replicates of the training sets D_{U1}^{*b}, *B* internal validation sets D_{U2}^{*b} and *B* testing sets D_{U3}^{*b}, then to re-estimate the model parameters $\hat{w}^{*b}$ on each training bootstrap sample q_{u1}^{*b}, to stop training on the basis of the associated validation bootstrap sample q_{u2}^{*b} and to test generalization performance, measured in terms of *KLIC*, on the test bootstrap sample q_{u3}^{*b}. In this bootstrap world, the empirical bootstrap distribution of the performance measure can be estimated, pseudo-errors can be computed, and used to approximate the distribution of the real errors. The approach is appealing, but characterized by very demanding computational intensity in real world contexts (see Fischer and Reismann 2002b for an application). Implementing the approach involves the following steps:

[18] This approach contrasts to residuals bootstrapping that treats the model residuals rather than $q_u=(x_u,\ y_u)$ as the sampling units and creates a bootstrap sample by adding residuals to the model fit. In this latter case bootstrapping distribution is conditional on the actual observations.

Step 1: Conduct totally independent re-sampling operations, where

(i) *B* independent training bootstrap samples are generated, by randomly sampling *U*1 times (*U*1<*U*), with replacement, from *D* for *b*=1, …, *B*

$$D_{U1}^{*b}=\left\{q_{u1}^{*b}=\left(x_{u1}^{*b},\ y_{u1}^{*b}\right)\right\}, \qquad (28)$$

(ii) *B* independent validation bootstrap samples are generated [in the case of early stopping only], by randomly sampling *U*2 times (*U*2<*U*), with replacement, from *D* so that for *b*=1, …, *B*

$$D_{U2}^{*b}=\left\{q_{u2}^{*b}=\left(x_{u2}^{*b},\ y_{u2}^{*b}\right)\right\}, \qquad (29)$$

(iii) *B* independent test bootstrap samples are generated, by randomly sampling *U*3 times (*U*3<*U*), with replacement, from *D* so that for *b*=1, …, *B*

$$D_{U3}^{*b}=\left\{q_{u3}^{*b}=\left(x_{u3}^{*b},\ y_{u3}^{*b}\right)\right\}. \qquad (30)$$

Step 2: Use each training bootstrap sample q_{u1}^{*b} to compute the bootstrap parameter estimates $\hat{w}^{*b}$ by solving Eq. (16) with q_{u1}^{*b} replacing q_u:

$$\hat{w}^{*b}=\arg\min\left\{\lambda\left(q_{u1}^{*b},\ w^{*b}\right):\ w^{*b}\in W\subseteq\mathbb{R}^{Q}\right\} \qquad (31)$$

where *Q* is the number of parameters, and

$$\lambda\left(q_{u1}^{*b},\ w^{*b}\right)=\sum_{u1=1}^{U1}\left[y_{u1}\ln\phi(w)_{u1}-\phi(w)_{u1}\right]. \qquad (32)$$

Note: During the training process the generalization performance of the model (in terms of the *KLIC* criterion) is monitored on the corresponding bootstrap validation set, in the case of early stopping. The training process is stopped if the validation error starts to increase.

Step 3: Calculate the *KLIC*-statistic $KLIC\,(D_{U3}^{*b})$ for each test bootstrap sample.

Step 4: Replicate Steps 3-4 many times, say *B*=100 or *B*=1,000.

Step 5: The statistical accuracy of the generalization performance statistic can then be evaluated by looking at the variability of the statistic between the different bootstrap test sets. Estimate the standard deviation $\hat{\sigma}$ of the statistic as approximated by bootstrap

$$\hat{\sigma}^B = \left\{ \frac{1}{B-1} \sum_{b=1}^{B} \left[\widehat{KLIC}^{*b}\left(D_{U3}^{*b}\right) - \widehat{KLIC}^{*}(\cdot) \right]^2 \right\}^{\frac{1}{2}} \tag{33}$$

with

$$\widehat{KLIC}^{*}(\cdot) = \sum_{b=1}^{B} \widehat{KLIC}^{*b}\left(D_{U3}^{*b}\right). \tag{34}$$

The true standard error of $\widehat{KLIC}$ is a function of the unknown density function, say F, of *KLIC*, that is $\sigma(F)$. With the bootstrapping approach described above one obtains $\hat{F}_{U3}^{*}$, which is supposed to describe closely the empirical distribution $\hat{F}_{U3}$, in other words $\hat{\sigma}_{U3}^{B} \approx \sigma(\hat{F}_{U3})$. Asymptotically, this means that the sample size tends to infinity, i.e. $U3 \to \infty$, the estimate $\hat{\sigma}^B$ tends to $\sigma(F)$. For finite sample sizes, however, there will be deviations in general.

8. CLOSING REMARKS

In this paper a modest attempt has been made to provide a unified framework for neural spatial interaction modelling, based upon maximum likelihood estimation under distributional assumptions of Poisson processes. In this way we avoid the weaknesses of least squares and normality assumptions that ignore the true integer nature of the origin-destination flows and approximate a discrete-valued process by an almost certainly misrepresentative continuous representation.

Randomness enters in two ways in neural spatial interaction modelling: in the splitting of the data set into training, validation and test sets on the one side, and in choices about parameter initialization on the other. The paper suggests the bootstrapping pairs approach to evaluate the performance, reliability and robustness of neural spatial interaction models. The approach is attractive, but computationally intensive.

Despite significant improvements in our understanding of the fundamentals and principles of neural spatial interaction modelling, there are many open problems and directions for future research. The design of a neural network approach suited to deal with the doubly constrained case of spatial interaction, for example, is still missing. Finding good global optimization procedures for solving the non-convex learning problems is still an important issue for further research even though some relevant work can be found in Fischer, Hlavácková-Schindler and Reismann (1999). Also the model identification problem deserves further attention to come up with techniques that go beyond the current rules of thumb.

From a spatial analytic perspective an important avenue for further investigation is the explicit incorporation of spatial dependency in the network representation that received less attention in the past than it deserves. Another is the application of Bayesian inference techniques to neural networks. A Bayesian approach would provide an alternative framework for dealing with issues of network complexity and would avoid many of the problems discussed in this paper. In particular, confidence intervals could easily be assigned to the predictors generated by neural spatial interaction models, without the need of bootstrapping.

REFERENCES

Baldi, P., Chauvin, Y., 1991. Temporal evolution of generalization during learning in linear networks. *Neural Computation*, 3(4), pp. 589-603

Bishop, C. M., 2006. *Pattern recognition and machine learning*. Springer, New York

Bishop. C. M., 1995. *Neural networks for pattern recognition*. Clarendon Press, Oxford

Cichocki, A., Unbehauen, R., 1993. *Neural networks for optimization and signal processing*. John Wiley, Chichester

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems*, 2, pp. 303-314

Efron, B., 1982. *The jackknife, the bootstrap and other resampling plans*. Society for Industrial and Applied Mathematics, Philadelphia [PA]

Finnoff, W., 1991. Complexity measures for classes of neural networks with variable weight bounds. In: *Proceedings of the International Geoscience and Remote Sensing Symposium* [IGARSS'94, Volume 4]. IEEE Press, Piscataway [NJ], pp. 1880-1882

Fischer, M. M., 2002. Learning in neural spatial interaction models: A statistical perspective. *Journal of Geographical Systems*, 4 (3), pp. 287-299

Fischer, M. M., 2001. Neural spatial interaction models. In Fischer M M, Leung Y (eds) GeoComputational modelling. Techniques and applications. Springer, Berlin, Heidelberg and New York, pp. 195-219

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 and New York, pp. 89-101

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

Fischer, M. M., Griffith, D. A., 2008. Modeling spatial autocorrelation in spatial interaction data: An application to patent citation data in the European Union. *Journal of Regional Science*, 48(5), pp. 969-989

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

Fischer, M. M., Reismann, M., 2002a. Evaluating neural spatial interaction modelling by bootstrapping. *Networks and Spatial Economics*, 2(3), pp. 255-268

Fischer, M. M., Reismann, M., 2002b. A methodology for neural spatial interaction modeling. *Geographical Analysis*, 34(2), pp. 207-228

Fischer, M. M., Hlavácková-Schindler, K., Reismann, M., 1999. A global search procedure for parameter estimation in neural spatial interaction modelling. *Papers in Regional Science*, 78(2), pp. 119-134

Fischer, M. M., Reismann, M., Hlavácková-Schindler, K., 2003. Neural network modelling of constrained spatial interaction flows: Design, estimation and performance issues. *Journal of Regional Science*, 43(1), pp. 35-61

Flowerdew, R., Aitken, M., 1982. A method of fitting the gravity model based on the Poisson distribution. *Journal of Regional Science*, 22(2), pp. 191-202

Fogel, D. B., 1995. *Evolutionary computation: Toward a new philosophy of machine intelligence*. IEEE Press, Piscataway [NJ]

Fotheringham, A. S., 1983. A new set of spatial interaction models: The theory of competing destinations. *Environment and Planning A*, 15(1), pp. 15-36

Funahashi, K., 1989. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3), pp. 183-192

Griffith, D. A., 2003. *Spatial autocorrelation and spatial filtering*. Springer, Berlin, Heidelberg and New York

Goldberg, D. E., 1989. *Genetic algorithms*. Addison-Wesley, Reading [MA]

Gopal, S., Fischer, M. M., 1997. Fuzzy ARTMAP – a neural classifier for multispectral image classification. In: Fischer, M. M., Getis, A. (eds) *Recent developments in spatial analysis*. Springer, Berlin, Heidelberg and New York, pp. 306-335

Hassoun, M. H., 1995. *Fundamentals of artificial neural networks*. MIT Press, Cambridge [MA] and London, England

Haykin, S., 1994. *Neural networks. A comprehensive foundation*. Macmillan College Publishing Company, New York

Hinton, G. E., 1987. Learning translation invariant recognition in massively parallel networks. In: Bakker, J. W. de, Nijman, A. J., Treleaven, P. C. (eds) *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*. Springer, Berlin, Heidelberg and New York, pp. 1-13

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), pp. 359-368

LeSage, J. P., Fischer, M. M., 2010. Spatial econometric modelling of origin-destination flows. In Fischer, M. M., Getis, A. (eds) *Handbook of Applied Spatial Analysis*. Springer, Berlin, Heidelberg and New York, pp. 409-433

LeSage, J. P., Pace, R. K., 2009. *Introduction to spatial econometrics*. CRC Press (Taylor and Francis Group), Boca Raton [FL], London and New York

Kullback, S., Leibler, R. A., 1951. On information and sufficiency. *Annals of Mathematical Statistics*, 22, pp. 78-86

McCulloch, W. S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, pp. 115-133

Moody, J. E., 1992. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In: Moody, J. E., Hanson, S. J., Lippman, R. P. (eds) *Advances in neural information processing systems* 4. Morgan Kaufmann, San Mateo [CA], pp. 683-690

Nocedal, J., Wright, S. J., 1999. *Numerical optimization*. Springer, Berlin, Heidelberg and New York

Press, W. H., Teukolky, S. A., Vetterling, W. T., Flannery, B. P., 1992. *Numerical recipes in C. The art of scientific computing*, 2nd edn. Cambridge University Press, Cambridge

Rosenblatt, F., 1962. *Principles of neurodynamics*. Spartan Books, Washington DC

Rumelhart, D. E., Durbin, R., Golden, R., Chauvin, Y., 1995. Backpropagation: The basic theory. In: Chauvin, Y., Rumelhart, D. E. (eds) *Backpropagation: Theory, architectures and applications*. Lawrence Erlbaum Associates, Hillsdale [NJ], pp. 1-34

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1986. Learning internal representations by error propagation. In: Rumelhart, D. E., McClelland, J. L., PDP Research Group (eds) *Parallel distributed processing: Explorations in the microstructure of cognition*. MIT Press Cambridge [MA], pp. 318-362

Sen, A., Smith, T. E., 1995. *Gravity models of spatial interaction behaviour*. Springer, Berlin, Heidelberg and New York

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society* B, 58, pp. 267-288

Weigend, A. S., Rumelhart, D. E., Huberman, B. A., 1991. Generalization by weight elimination with application to forecasting. In: Lippman, R., Moody, J., Touretzky, D. (eds) *Advances in neural information processing systems* 3. Morgan Kaufmann, San Mateo [CA], pp. 875-882

Weng, J., Hwang, W.-S., 2006. From neural networks to the brain: Autonomous mental development. *IEEE Computational Intelligence Magazine*, 1(3), pp. 15-31

Wilson, A. G., 1970. *Entropy in urban and regional planning*. Pion, London

Wilson, A. G., 1967. A statistical theory of spatial distribution models. *Transportation Research*, 1, pp. 253-269

Zapranis, A., Refenes, A.-P., 1999. *Principles of neural model identification, selection and adequacy. With applications to financial econometrics*. Springer, London