WSG Diskussionspapiere

WSG 59/97

Computational Neural Networks: A New Paradigm for Spatial Analysis

Manfred M. Fischer

Institut für Wirtschafts- und Sozialgeographie
Wirtschaftsuniversität Wien

Department of Economic and Social Geography
Vienna University of Economics and Business Administration

WSG Discussion Papers

Abteilung für Theoretische und Angewandte Wirtschafts- und Sozialgeographie
Institut für Wirtschafts- und Sozialgeographie
Wirtschaftsuniversität Wien

Vorstand: o.Univ.Prof. Dr. Manfred M. Fischer
A - 1090 Wien, Augasse 2-6, Tel. (0222) 313 36 - 4836

Redaktion: Univ.Ass. Mag.Dr. Petra Staufer

WSG 59/97

Computational Neural Networks: A New Paradigm for Spatial Analysis

Manfred M. Fischer

WSG-Discussion Paper 59

September 1997

Gedruckt mit Unterstützung
des Bundesministerium
für Wissenschaft und Verkehr
in Wien

WSG Discussion Papers are interim
reports presenting work in progress
and papers which have been submitted
for publication elsewhere.

ISBN 3 85037 070 4

1. Introduction

Spatial analysis (SA) represents one of the currently most valuable geographical assets that has a number of practical applications outside of geography. Examples include strategic land use and transportation planning, environmental analysis and planning, and allocation for service provision such as education, health and policing. Increasingly market companies, major retailers, supermarket chains, environmental consulting companies, car dealers and oil companies are starting to use geographic information and SA-tools - often within a Geographic Information System (GIS) environment - to understand their market better and to identify niches [see, e.g., Burrough et al. 1997, Birkin, Clarke and George 1995].

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

Most of the spatial data analysis techniques and methods currently in use were developed in the 1960s and 1970s, i.e. in an era of scarce computing power and small data sets. Their current implementations take only limited advantage of the data storage and retrieval capabilities of modern computational techniques, and basically ignore both the emerging new era of parallel supercomputing and the computational intelligence techniques. In addition, they overemphasize linear statistical model designs while non-linearities prevail in reality, tend to neglect rather than take into account the special nature of spatial data, and exhibit major difficulties to cope with the information rich environments on which societies and economies increasingly depend.

No doubt, spatial analysis in general and spatial data analysis in particular is currently entering a period of rapid change, a period which presents the unique opportunity for novel styles of data analysis in order to meet the new needs for efficiently and comprehensively exploring large databases for patterns and relationships against a background of data uncertainty and noise, especially when the underlying database is of the order of gigabytes. The paper intends to provide a systematic introduction to computational neural networks and is designed to help spatial analysts to learn about this exciting new field. My second goal is to illustrate the power of computational neural network viz-à-viz conventional modelling and techniques in one application field with noisy data of limited record length: *spatial interaction modelling* which has a special significance in the historical development of mathematical and statistical models in geography and regional science as the testing ground for novel approaches. My hope is that this contribution will help to demystify and popularize computational neural networks in spatial analysis and stimulate methodologically and technically sound, rather than adhoc neural network applications in spatial analysis.

The paper is structured as follows. Section 2 briefly summarizes the computational appeal of neural networks for solving some fundamental spatial analysis problems, followed by a definition of computational neural network models in mathematical terms [section 3]. Sections 4 to 6 are devoted to a discussion of the three definitional components of a computational neural network: properties of the processing elements [section 4], network topology [section 5] and learning [determination of the weights on a connection] in a computational neural network [section 6]. Section 7 presents a taxonomy of computational neural networks. It breaks neural networks down according to the topology and type of interconnections [network topology] and the learning [training] paradigm adopted. Section 8 illustrates the attractiveness of computational neural network models - against the conventional modelling approach of the gravity type - for spatial interaction modelling. Some conclusions and an outlook are provided in the final section.

2. Why Computational Neural Networks?

Briefly stated, a computational neural network model - simply CNN model - is a parallel, distributed information structure consisting of a set of adaptive processing (computational) elements and a set of unidirectional data connections (a more elaborated definition is given below). These networks are **neural** in the sense that they have been inspired by neuroscience but not necessarily because they are faithful models of biological neural or cognitive phenomena. In fact, the networks covered in this paper are more closely related to traditional mathematical and/or statistical models such as non-parametric pattern classifiers, statistical regression models and clustering algorithms than they are to neurobiological models.

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

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

The strongest appeal of CNNs is their suitability for *machine learning* [i.e computational adaptivity]. Machine learning in CNNs consists of adjusting connection weights to improve performance of a CNN. This is a very simple and pleasing formulation of the learning problem.

Certainly *speed of computation* is another key attraction. In traditional single processor Von Neuman computers, the speed is limited by the propagation delay of the transistors. Computational neural networks, on the other hand, because of their intrinsic massively parallel distributed structure, can perform computations at a much higher rate, especially when these CNNs are implemented on parallel digital computers, or, ultimately, when implemented in customized hardware. This feature makes it possible to use CNNs as tools for real-time applications involving, for example, pattern recognition in GIS- and Remote Sensing (RS)-data environments. It is clear that the increasing availability of parallel hardware will enhance the attractiveness of CNNs and other basically parallel models in spatial analysis. Furthermore, because of their non-linear nature, CNNs can perform functional approximation and pattern classification operations, which are beyond optimal linear techniques. CNNs offer a *greater representational flexibility* and *freedom from linear model design*. They are semi- or non-parametric and make weaker assumptions concerning the shape of underlying distributions than conventional statistical models. Other important features include *robust behaviour* with noisy data. Noise refers to the probabilistic introduction of errors into data. This is an important aspect of real-world aplications, and CNNs can be especially good at handling noise in a reasonable manner.

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

❒ **Pattern classification**

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

❒ **Clustering/Categorization**

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

❒ **Function Approximation**

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

❒ **Prediction/Forecasting**

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

❒ **Optimization**

A wide variety of problems in spatial analysis can be posed as [non-linear] spatial optimization problems. The goal of an optimization algorithm is to find a solution satisfying a set of constraints such that an objective function is maximized or minimized. The Travelling salesman Problem, an NP-complete problem, and the problem of finding optimal locations are classical examples.

3. Definition of a Computational Neural Network

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

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

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

For illustration purposes a typical CNN architecture is pictured in figure 1. In the figure the circles denote the processing elements, and the arrows the direction of the signal flow. The single output signal of the PEs branches into copies which are distributed to other PEs or which leave the network altogether. The input to the network presented by the external world can be viewed as data array $\mathbf{x}=(x_1,...,x_n) \in \Re^n$ [n-dimensional euclidean space] and the output of the network as data array $\mathbf{y}=(y_1,...,y_m) \in \Re^m$ [m-dimensional euclidean space with m<n]. When viewed in this manner, the CNN can be thought of as a function $\Phi: \Re^n \rightarrow \Re^m$.

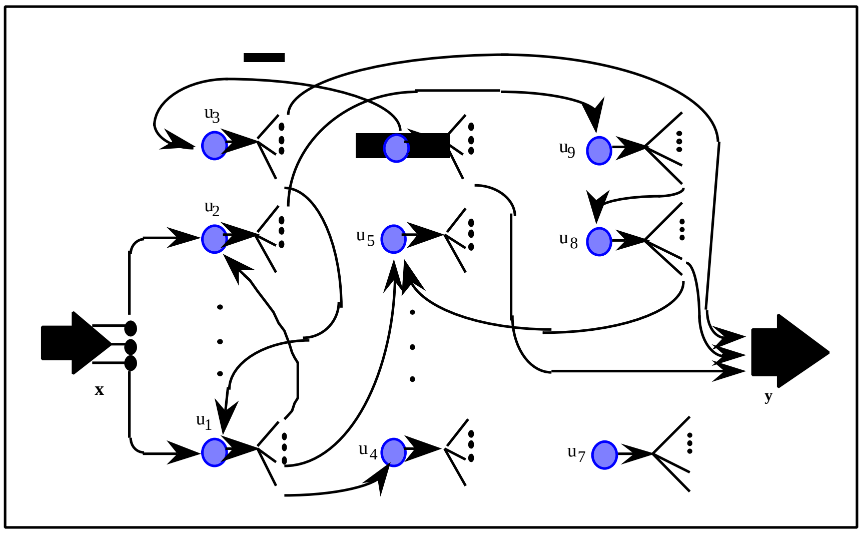


Figure 1: A typical CNN architecture

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

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

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

4. Properties of the Processing Elements

Processing elements are fundamental to the operation of any CNN. Thus, it is important to have a closer look at the operation of such processing elements that form the basis of the CNNs considered in this paper.

Figure 2 shows internal details of a generic processing unit i from an arbitrary computational neural network. The processing unit in this figure occupies a position in its network that is quite general; i.e. this PE both accepts inputs from other PEs and send its output (activation) to other PEs. PEs that are neither input nor output units maintain this generality and function as fundamental non-linear computing devices in a CNN.

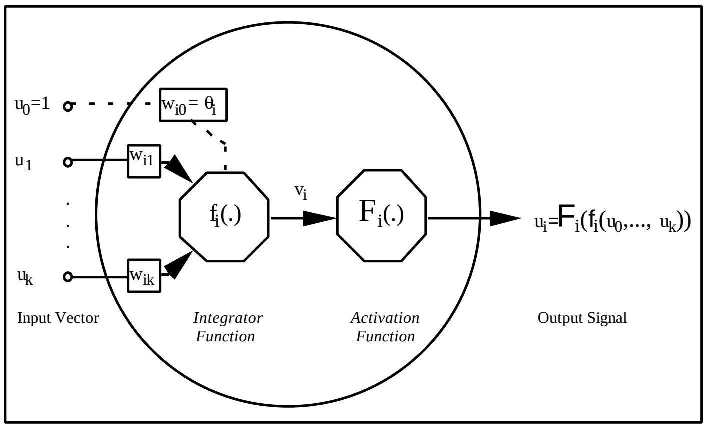


Figure 2: Generic processing element u_i

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

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

Positive weights indicate reinforcement, negative weights represent inhibition. By convention there is a unit u_0 whose output is always +1 that is connected to the PE in question. The corresponding weight w_{i0} is called bias θ_i of unit i (i.e. the bias is treated as any other weight). Thus, we may define the (k+1)-by-1 input vector

$$\mathbf{u}=[1, u_1, u_2,..., u_k]^T. \tag{1}$$

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

$$\boldsymbol{W}=[\theta, w_{i1},..., w_{ik}]^T. \tag{2}$$

The basic operation of a processing element to compute its activation or output signal u_i involves applying a transfer function φ_i that is composed of two mathematical functions [Fischer 1995]: an integrator function f_i and an activation or output function F_i, i.e.

$$u_i = \varphi_i(\mathbf{u}) = \mathsf{F}_i\,(f_i\;(\mathbf{u})) \tag{3}$$

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

The **integrator function** f_i performs the task to integrate the activations of units directly connected to the processing element in question and the corresponding weights for those connections and, thus, to reduce the k arguments to a single value (called net input to or activity potential of the PE) v_i. Generally, f_i is specified as the inner product of the vectors **u** and ***W***, as follows

$$v_i = f_i(\mathbf{u}) = \langle \mathbf{u}, \boldsymbol{W} \rangle = \sum_{j=0,1,...,k} w_{ij} u_j \tag{4}$$

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

The activation or output function, denoted by $\mathcal{F}_i(f\ (.))$, defines the output of the processing unit in terms of the net input v_i at its input. There are various possibilities to specify $\mathcal{F}_i$. But usually $\mathcal{F}_i$ is specified as a non-linear, non-decreasing, bounded (i.e. $\mathcal{F}_i$ is expected to approach finite maximum values asymptotically) and piece-wise differentiable functional form. For computational efficiency it is desirable that its derivative be easy to compute.

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

$$F_i(v_i) = \frac{1}{1 + \exp(-\beta v_i)} \tag{5}$$

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

5. Network Topologies

In the previous section we discussed the characteristics of the basic processing element in a computational neural network. This section focuses on the pattern of connections between the PEs [termed architecture] and the propagation of data. As for the pattern of connection a major distinction can be made between

- **feedforward** computational neural networks, and
- **recurrent** computational neural networks.

Feedforward computational neural networks are networks that do not contain directed cycles. It is often convenient to organize the nodes of feedforward CNN into layers. We define a l-layer feedforward network as a CNN where processing units are grouped into l+1 subsets (layers) L_0, L_1,..., L_l such that if u in a layer L_a is connected to cell u_i in a layer L_b then a<b. For a strictly l-layer network we require additionally that units can be connected only to units in the next layer, i.e. b=a+1. All units in a layer L_0 are input units, all trainable units are in layers L_1,..., L_l, and all units in layer L_l are output units.

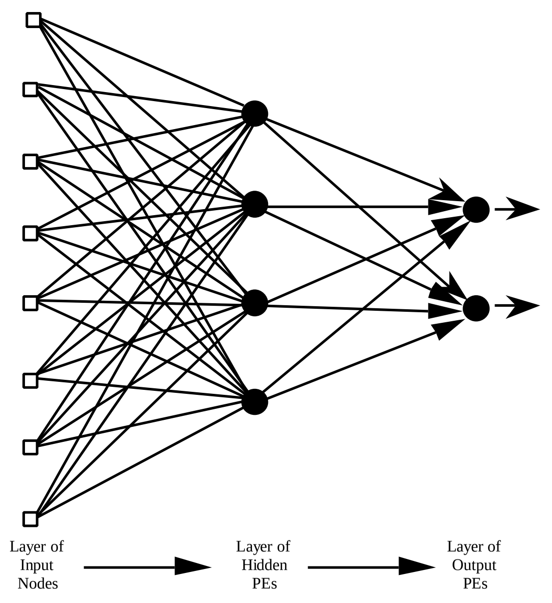
In the simplest form of a layered feedforward CNN, we just have a layer L_0 of input nodes that projects on to a layer L_1 of output computational units. Such a CNN is called a **single-layer feedforward network**, with the designation 'single-layer' referring to the output layer of PEs. In other words,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

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

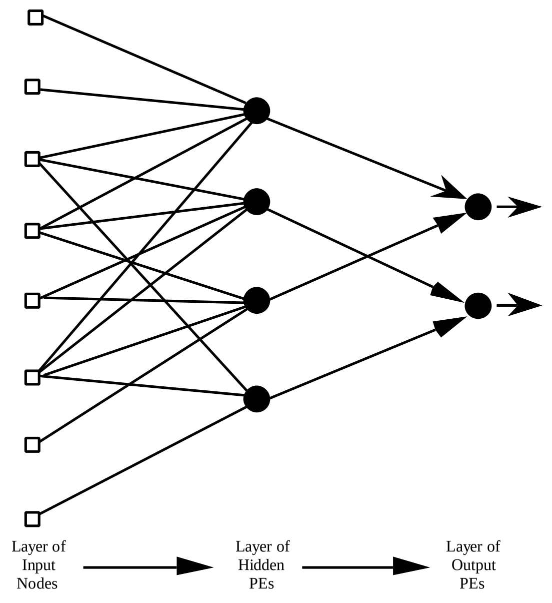
The nodes of the input layer L_0 supply the respective elements of the input vector which constitute the input signals applied to the PEs of the second layer [i.e. the first hidden layer L_1]. The output signals of L_1 are used as the inputs to L_2, and so on for the rest of the network. Characteristically the PEs in each layer are assumed to have the same transfer function φ and to have as their inputs the output signals of the preceeding layer. The set of output signals of the PEs in the output layer L_l constitutes the overall response of the CNN.

The architectural graphs of figure 3 illustrate the layout of **multilayer feedforward CNNs** for the case of a single hidden layer, so-called single hidden layer feedforward networks. For brevity the networks of figure 3 are referred to as 8:4:2 networks in that they have 8 input nodes, 4 hidden PEs and 2 output units. The CNN of figure 3(a) is said to be **fully connected** in the sense that every node of the network is connected to every other node in the adjacent forward layer. If some of the connections are missing from the network, then the network is said to be **partially connected**. An example of such a CNN is displayed in figure 3(b). Each PE in the hidden layer is connected to a partial set of input nodes in its immediate neighborhood. Such a set of localized nodes feeding a PE is said to constitute the **receptive field** of the PE. The CNN of figure 3(b) has the same number of input nodes, hidden units and output units as that of figure 3(a). But it has a specialized structure. In real world applications, the specialized structure built into the design of a feedforward CNN reflects **prior information** about the problem to be analyzed.

The procedure for translating a feedforward CNN diagram as, e.g. , illustrated in figure 3, into the corresponding mathematical function Φ follows from a straightforward extension of



(a) Fully connected CNN



(b) Partially connected CNN

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

section 3. The output of unit i is obtained by transforming the net input (see equation 4) with a non-linear activation function F_i to give

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

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

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

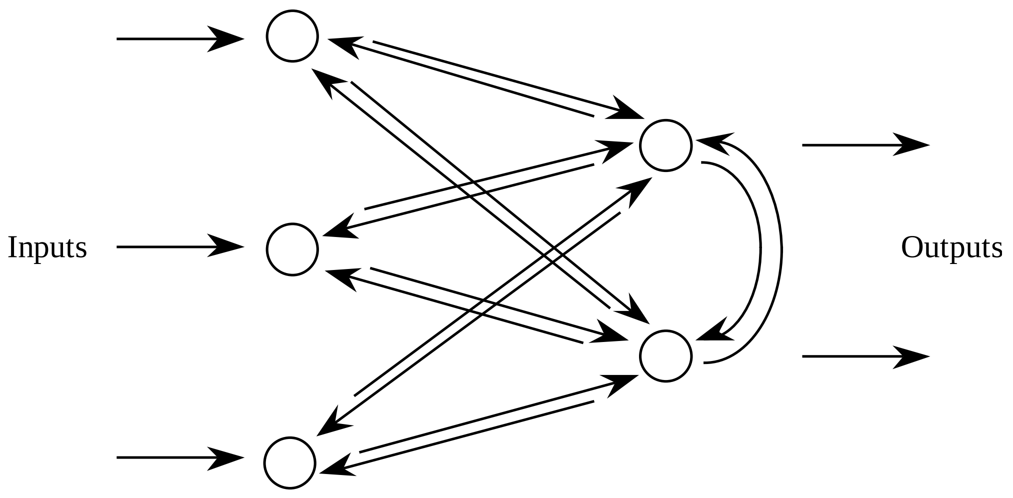


Figure 4: A recurrent computational neural network architecture

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

6. Learning in a Computational Neural Network

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

Each CNN is associated with one or more algorithms for machine learning. The input to these algorithms consists of a finite set of training examples, also called training patterns, {$\boldsymbol{E}^r$}. {$\boldsymbol{E}^r$} is a n-vector of values that gives settings for the corresponding units of a CNN model, as illustrated in figure 6. The j-th component of an example, $\boldsymbol{E}_j^r$ is assigned to input unit u_j (j=1,..., n).

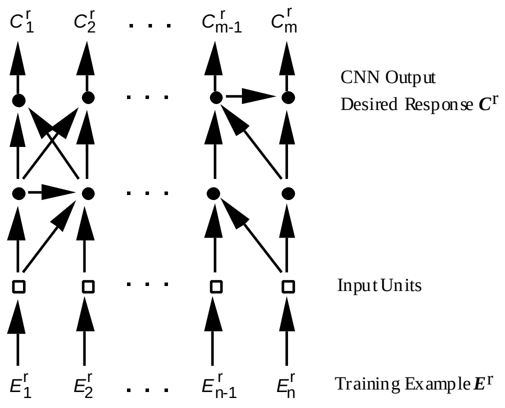


Figure 5: Training example inputs and outputs for a supervised learning model

It is convenient to distinguish two types of learning situations:

❐ **first**, supervised learning problems, and

❒ **second**, unsupervised learning problems.

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

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

There is a wide variety of learning algorithms available for solving both supervised and unsupervised learning problems that have been designed for specific network architectures. Many learning algorithms - especially those for supervised learning in feedforward CNNs - have their roots in function-minimization algorithms that can be classified as local or global search [minimization] algorithms. Learning algorithms are termed local if the computations needed to update each weight of a CNN can be performed using information available locally to that weight. This requirement may be motivated by the desire to implement learning algorithms in parallel hardware. Local minimization algorithms, such as those based on gradient-descent, conjugate-gradient and quasi-Newton methods (see Fischer and Staufer 1997), are fast but usually converge to local minima. In contrast, global minimization algorithms, such as simulated annealing and evolutionary algorithms, have heuristic strategies to help escape from local minima. All these algorithms are weak in either their local or their global search. For example, gradient information useful in local search is not used well in simulated annealing and evolutionary algorithms. In contrast, gradient-descent algorithms with multistarts are weak in global search.

Designing efficient algorithms for CNN learning is a very active research topic. In formulating CNN solutions for [large-scaled] real world problems, we seek to minimize the resulting algorithmic complexity that refers to the time required for a learning algorithm to estimate a solution from training patterns.

7. A Taxonomy of Computational Neural Networks

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

Backpropagation computational neural networks have emerged as major workhorses in spatial analysis. They can be used as universal function approximators in areas such as spatial regression, spatial interaction, spatial choice and space-time series analysis as well as pattern classifiers in data-rich environments [see, e.g., Openshaw 1993, Fischer and Gopal 1994, Fischer et al. 1994]. Strictly speaking, backpropagation is a technique which provides a computationally efficient procedure for evaluating the derivatives of the network's performance function with respect to the network parameters and corresponds to a propagation of errors backwards through the network. This technique was popularized by Rumelhart, Hinton and Williams (1986).

But backpropagation is used primarily with multi-layer feedforward networks [also termed multi-layer perceptrons], so it is convenient to refer to this type of supervised feedforward network as a backpropagation network. Backpropagation CNNs may be characterized by the following network, processing element and learning properties:

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

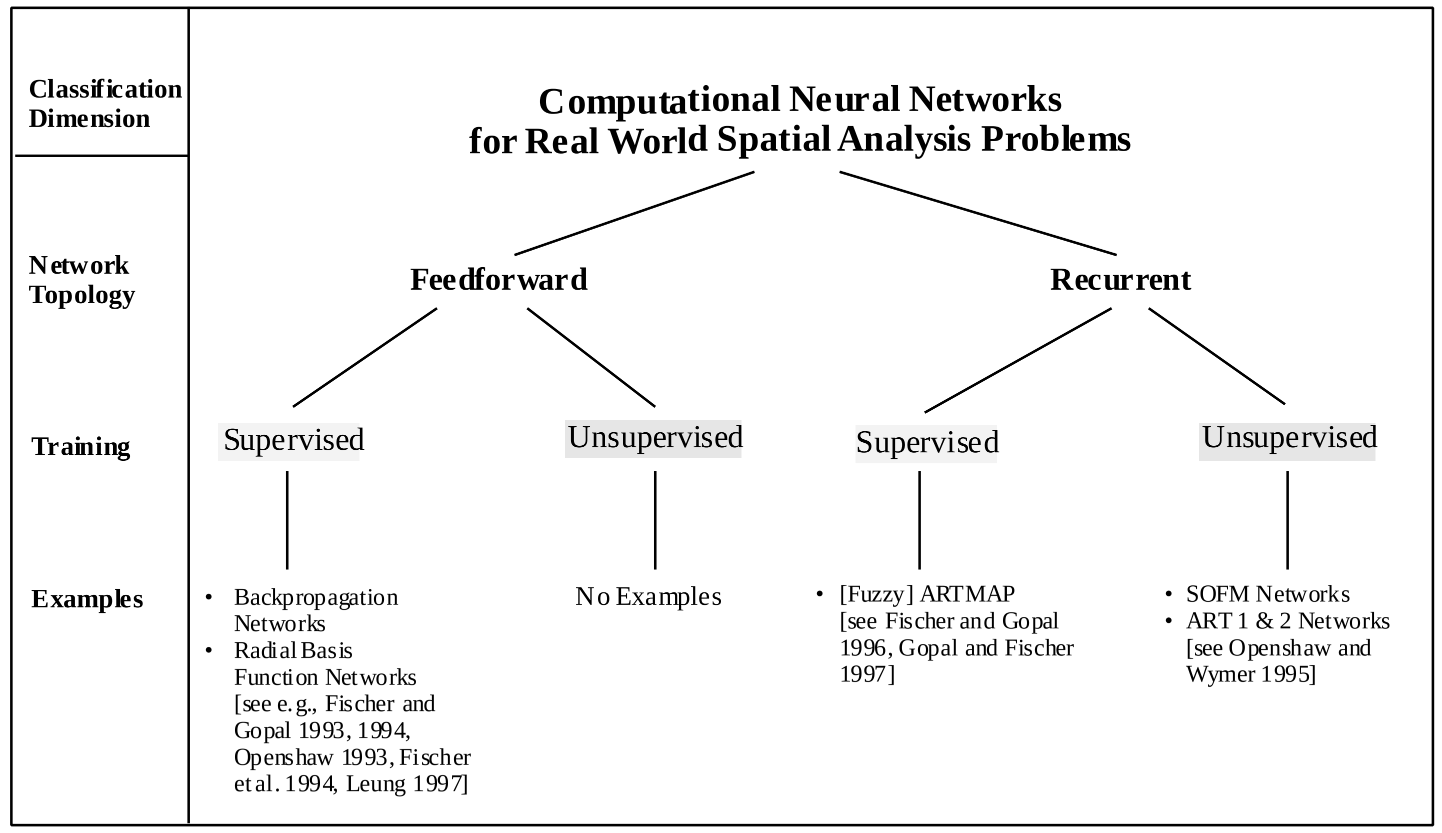

Classification Dimension
Network Topology
Training
Examples
Computational Neural Networks for Real World Spatial Analysis Problems
Feedforward
Recurrent
Supervised
Unsupervised
Supervised
Unsupervised
• Backpropagation Networks
• Radial Basis Function Networks [see e.g., Fischer and Gopal 1993, 1994, Openshaw 1993, Fischer et al. 1994, Leung 1997]
No Examples
• [Fuzzy] ARTMAP [see Fischer and Gopal 1996, Gopal and Fischer 1997]
• SOFM Networks
• ART 1 & 2 Networks [see Openshaw and Wymer 1995]

Figure 6: A taxonomy of computational neural networks

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

The **Radial Basis Function CNNs** are a special class of a single hidden layer feedforward network. Each processing element in the hidden layer utilizes a **radial basis function**, such as a Gaussian kernel, as the transfer function. The argument of the transfer function of each hidden unit computes the Euclidean norm between the input vector and the centre of the unit. The kernel function is centered at the point specified by the weight vector associated with the PE. Both the positions and the widths of these kernels must be learned from training patterns. Each output unit implements characteristically a *linear* combination of these radial basis functions. From the point of view of function approximation, the hidden units provide a set of functions that constitute a basis set for representing input patterns in the space spanned by the hidden units. There is a variety of learning algorithms for the Radial Basis Function CNNs. The basic one utilizes hybrid learning that decouples learning at the hidden layer from that at the output layer. This technique estimates kernel positions and kernel widths using a simple unsupervised k-means based clustering algorithm, followed by a supervised least mean square algorithm to determine the connection weights between the hidden and the output layer. Because the output units are typically linear, a non-iterative algorithm can be used. After this initial solution is obtained, a supervised gradient-based algorithm can be utilized in a further step to refine the connection parameters. It is worthwhile to note that Radial Basis Function Networks require more training data and more hidden units compared with backpropagation networks for achieving the same level of accuracy, but train by orders of magnitude faster.

ART network models differ from the previous CNNs in that they are recurrent. The data are not only fed forward but also back from the output to the input units. ART networks, moreover, are biologically motivated and were developed as possible models of cognitive phenomena in humans and animals. The basic principles of the underlying theory of these networks, known as **adaptive resonance theory** (ART), were introduced by Grossberg (1976a, b). They are essentially clustering CNNs with the task to automatically group unlabeled input vectors into several categories (clusters) so that each input is assigned a label corresponding to a unique cluster. The clustering process is driven by a similarity measure, vectors in the same cluster are similar which means that they are close to each other in the input space. The networks use a simple representation in which each cluster is represented by the weight vector of a prototype unit. If an input vector is close to a prototype, then it is considered a member of the prototype's cluster, and differences are attributed to unimportant features or to noise. The input and the stored prototype are said to **resonate** when they are sufficiently similar. There is no set number of clusters, clusters are created as needed. Any clustering algorithm that does not prespecify the number of clusters must have some parameter that controls cluster granularity. For ART models this parameter is called the **vigilance parameter**. Regardless of its setting the ART networks are stable for a finite set of training examples in that final clusters will not change with additional iterations from the original set of training examples. Thus, they show incremental clustering capabilities and can handle an infinite stream of input data. They do not require large memory to store training data because their cluster prototype units contain implicit representation of all the inputs previously encountered. However, ART networks are sensitive to the presentation order of the training examples. They may yield different clustering on the same data when the presentation order of patterns is varied. Similar effects are also present in incremental versions of classical clustering techniques such as k-means clustering [i.e., k-means is also sensitive to the initial choice of cluster centres].

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

❒ an input processing field (called F_1-layer) consisting of two parts: the input portion with input nodes and the interface portion [interconnection],

❒ the layer of linear units (called F_2-layer) representing prototype vectors whose outputs are acted as on competitive learning [i.e. the winner is the node with the weight vector closest to the input vector, closest in a euclidean distance sense], and

❒ supplemental units implementing a reset mechanism to control the degree of similarity of patterns placed on the same cluster.

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

ART models come in several varieties, most of which are unsupervised, and the simplest are ART 1 designed for clustering training examples $\{\boldsymbol{E}^r\}$ with discrete data, $\boldsymbol{E}_j^r \in \{0,1\}$ [Carpenter and Grossberg 1987a] and ART 2 designed for continuous data $\boldsymbol{E}_j^r \in [0,1]$ [Carpenter and Grossberg 1987b]. A more recent addition to the ART family is ARTMAP, a supervised learning version of ART, consisting of a pair of ART1 modules for binary training and teaching signals linked together via an inter-ART associative memory, called a map field [Carpenter, Grossberg and Reynolds 1991a]. Fuzzy ARTMAP is a direct generalization of ARTMAP for continuous data which is achieved by replacing ART 1 in ARTMAP with fuzzy ART [Carpenter, Grossberg and Rosen 1991b]. Fuzzy ART synthesizes fuzzy logic and adaptive resonance theory by exploiting the formal similarity between the computations of fuzzy subsethood and the dynamics of prototype choice, search and learning. This approach is appealing because it nicely integrates clustering with supervised learning on the one side and fuzzy logic and adaptive resonance theory on the other. Recently Gopal and Fischer (1997) illustrated fuzzy ARTMAP performance in relation to that of backpropagation and the maximum likelihood classifier on a real world setting of a spectral pattern recognition problem (see also Fischer and Gopal 1996).

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

The essential characteristics of such networks may be summarized as follows:

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

❐ **processing element properties**: each output unit is charcterized by a n-dimensional weight vector; linear processing elements because each Kohonen unit computes its net input linearly with respect to its inputs; non-linearities come in when the selection is made as to which unit ‘fires’,

❒ **learning properties**: unsupervised learning that consists of adaptively modifying the connection weights of a network of locally interacting units in response to input excitations and in accordance with a competitive learning rule [i.e. weight adjustment of the winner and its neighbours]; the weight adjustment of the neighbouring PEs is instrumental in preserving the order of the input space.

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

It is hoped that the different families of CNN models presented in this section give the reader an appreciation of the diversity and richness of these models. In the remainder of this paper we will be working with feedforward CNNs in general and backpropagation CNNs in particular. There is no doubt that these models provide spatial analysts an extremely rich class of valuable non-linear mathematical tools. Their application to real world settings holds the potential for fundamental advances in empirical understanding across a broad spectrum of application fields in spatial analysis.

8. Applications of Backpropagation Networks as Function Approximators in a Spatial Context

We have discussed a number of important CNN models which may be used for solving the five classes of spatial analysis problems described in section 2. To successfully work with real-world problems, one must deal with numerous design issues, including network model, network size, activation functions, learning parameters, and a number of training samples. This section briefly describes a spatial interaction example and relies on Fischer and Gopal (1994) to illustrate the behaviour of backpropagation CNNs as function approximators in spatial analysis.

Backpropagation CNNs are currently the most important and most widely used networks for function approximation in a spatial analysis context, particularly for spatial regression or spatial interaction modelling [see, e.g., Openshaw 1993, Fischer and Gopal 1994, Gopal and Fischer 1996]. This is the reason why we briefly illustrate the application of this model type in the context of modelling interregional telecommunication flows and evaluate its performance viz-à-viz a conventional benchmark model. The telecommunication data used stem from network measurement of carried traffic in Austria in 1991 [measured in in terms of erlang] and refer to the traffic between the 32 telecommunication districts in Austria. The data set for the model building process is made up of a set of 992 4-tuples (A_r, B_s , D_{rs},T_{rs}^{obs}) with r, s = 1,..., 32 and r $\neq$ s, where the first three components represent the intensity of telecommunication generated by region r [measure used: gross regional product], destination-specific pull factors of region s [measure used: gross regional product], and the inhibiting effect of geographic separation between region r and s [measure used: distance]. The last component refers to the observed measurement, the target or output value. Due to measurement problems, intraregional traffic is left out of consideration. The log-normal version of the unconstrained spatial interaction model is used as **benchmark model**.

Data preprocessing is often one of the most important steps for the success of CNN modelling in real world contexts. In the simplest case, preprocessing takes the form of a linear transformation of the input data, and also the output [target] data which is sometimes called post-processing. In this study the input and output data were preprocessed in this application context to logarithmically transformed data scaled into [0.1, 0.9], due to two reasons: first, to make the comparison with the unconstrained log-normal version of the gravity model as close as possible, and, second, because a logistic PE always produces an activation strictly greater than 0 and strictly less than 1. This transformation procedure keeps targets in the quasi-linear part of the sigmoid where learning is faster.

The backpropagation CNN used in this application is defined by the following specifications:

❒ **Network Topology**: Fully connected single hidden feedforward network with three input units representing A_r, B_s , D_{rs}; 30 intermediate units and one output unit denoting the derived intensity of telecommunication from region r to region s.

❒ **Processing Unit Properties**: Inputs and activations are continuous, assuming values on [0, 1], choice of logistic PE activation functions [see equation (5)].

❒ **Model Equation**: In mathematical terms the CNN model may be written in a compact form as

$$\mathbf{y} = \Phi(\mathbf{x},\mathbf{W}) = \left\{1 + \exp\left[-\sum_{h=0}^{30} \alpha_h \left[1 + \exp\left(-\sum_{i=0}^{3} \beta_{hi} x_i\right)\right]^{-1}\right]\right\}^{-1} \tag{7}$$

where **x** is the 3-dimensional input vector, **y** the 1-dimensional output vector, Φ the network function [that is a composite function from input to output space], **W** the vector of all adjustable weights, α_h the weight on the connection from the h-th hidden to the output unit and β_{hi} the weight on the connection from the i-th input to the h-th hidden unit. Note that the method of cross-validation had been utilized to determine the size of the network model and arrived at 30 hidden units.

❒ **Learning Algorithm**: The backpropagation technique combined with an epoch-based stochastic version of the gradient descent technique [epoch size of 20 input patterns presented in random order] is used to adjust the network weights so as to minimize the familiar mean square error function

$$\mathrm{J}(\mathbf{W}) = \tfrac{1}{2}\left\| C_t^r - \Phi(E_t^r, \mathbf{W}) \right\|^2 \tag{8}$$

which is evidently continuous and differentiable with respect to the components of the weight vector **W**. (E_t^r, C_t^r) denotes a sequence of input-output training patterns. J(**W**) guides the learning process in a feedback error-correction manner [backpropagation technique] as the system performs stoachastic gradient descent on the unknown mean square error surface.

❒ **Initial weights**: To start the iteration process initial weights were randomly generated from a uniform distribution between -0.1 and 0.1. Five different random initializations were used to analyze variations due to different random initial conditions as well as in the random sequence of the input patterns.

❒ **Step Size**: The gradient descent algorithm contains a constant, called learning rate parameter, that governs how big a step is made at each iteration. Unfortunately the choice of this parameter is problem-specific and can greatly affect the operation of the algorithm. Typically the learning rate can range from 0.001 to 0.2. Values for the learning rate that are too large cause the system to become unstable where the magnitude of the weight values oscillate and diverge. Values that are too small cause the system to learn too slowly. In this application context a learning rate of 0.15 has been chosen. In order to speed convergence while avoiding instability a momentum term of 0.8 has been added to the weight adjustment.

❒ **Performance Measures**: Model performance is measured by means of two performance measures: the first is the average relative variance of a set of 20 randomly chosen patterns, a widely used measure in the NN-literature [see Weigend, Rumelhart and Hubermann 1991]:

$$\mathrm{ARV}(\mathrm{S}) = \frac{\sum_{\mathrm{t}\in \mathrm{S}} \left\| C_{\mathrm{t}}^{\mathrm{r}} - \mathrm{F}\,(E_{\mathrm{t}}^{\mathrm{r}}, \mathbf{W}) \right\|^2}{\sum_{\mathrm{t}\in \mathrm{S}} \left\| C_{\mathrm{t}}^{\mathrm{r}} - \overline{C}^{\mathrm{r}} \right\|^2} \tag{9}$$

where $C_{\mathrm{t}}^{\mathrm{r}}$ denotes the r-th component of the desired output associated with input $E_{\mathrm{t}}^{\mathrm{r}}$ and $\overline{C}^{\mathrm{r}}$ is the average value of $C_{\mathrm{t}}^{\mathrm{r}}$. The second performance measure used is the coefficient of determination $\mathrm{R}^2(\mathrm{S})$, a widely used goodness-of-fit measure in spatial interaction modelling:

$$\mathrm{R}^2(\mathrm{S}) = \frac{\sum_{\mathrm{t}\in \mathrm{S}} \left\| \Phi(E_{\mathrm{t}}^{\mathrm{r}}, \mathbf{W}) - \overline{C}^{\mathrm{r}} \right\|^2}{\sum_{\mathrm{t}\in \mathrm{S}} \left\| C_{\mathrm{t}}^{\mathrm{r}} - \overline{C}^{\mathrm{r}} \right\|^2} \tag{10}$$

It is important to emphasize that the goal of network training is not to learn an exact representation of the training data itself, but rather to build a mathematical model of the process that generates the data. This is of crucial importance - but generally neglected so far - if the CNN model is to exhibit good generalization, that is, to make good predictions for new inputs. We can obtain a better fit of the CNN to the training data by increasing the number of degrees of freedom [i.e. the number of free parameters] in the CNN model which gives it greater flexibility. If, however, we increase the model complexity too much, then the approximation to the underlying function actually gets worse. Such functions are said to be over-fitted to the data. The point of best generalization in feedforward CNN models is determined by the trade-off between the bias and the variance of the model, and arises when the number of degrees of freedom in the CNN model is relatively small compared to the size of the data set. There are several techniques to control overfitting of a CNN model: **regularization techniques**, **network pruning techniques** [see, e.g., Fischer et al. 1994] and **cross-validation**.

Cross validation has been used in the study reported here. Various feedforward CNNs have been trained by minimization the one-half sum-of-square error function defined by (8) with respect to a **training data set** [50% of all the data available in this application]. The performance of the CNNs then has been compared by evaluating the performance criterion using an independent **validation data set** [15% of all the available data], and the network having the smallest error with respect to the validation data set chosen. This approach is called the hold cut method. Since this procedure can itself lead to some over-fitting to the validation data set, the performance of the selected network should be confirmed by measuring its perfomance on a third independent set of data [35% of the total set], called prediction or test set.

Figure 7 shows the performance of the CNN model as a function of training time epochs with an epoch size of 20 patterns . The average relative variances are given for the training set, the validation set and the prediction set. The ARV error of the CNN measured using the training set continuously decreases and seems to level out after 5,000 epochs. This is what one expects. The validation test set error as shown in figure 5 decreases first, after 1,500 epochs only at a moderate rate with 4,250, then slightly increases and tends to approach an asymptotic value. If we assume that the error curve of the CNN model tested against the entire infinite set of possible patterns would be approximately the same as that of the validation set curve which is only a crude assumption, then clearly we would like to stop training when this curve arrives at its minimum. The minimum is reached after 4,250 epochs. At this stopping point P, the model is used for prediction. In the specific choice of a training set-validation set combination shown in figure 7, the fitting of the noise of the training set occurs to have only a little effect on the error of the validation set which is also reflected in the prediction set curve.

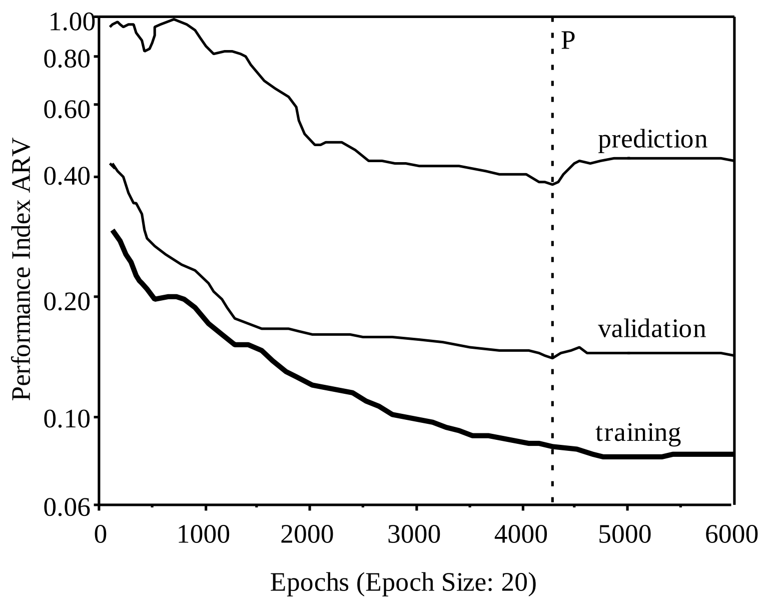


Figure 7 : Training, validation and prediction set curves of the CNN model as a function of training time in epochs (the vertical line P indicates the stopping point)

Table 1 reports the prediction performance of the CNN model on the test set in terms of ARV and R^2 averaged over five trials that differ in initial conditions and in the random sequence of the input data. In order to make the comparison with the benchmark model as close as possible, this model was estimated (tested) with the same data seen by the CNN during training (testing). One can see that the CNN outperforms the conventional model in all trials.

Table 1: Testing performance of the neural net and the conventional model*

	Neural Net Model		**Conventional Model**	
	ARV	**R^2**	**ARV**	**R^2**
Prediction (Testing)				
Trial 1	0.4063	0.5937	0.4630	0.5431
Trial 2	0.4057	0.5942	0.4611	0.5390
Trial 3	0.4063	0.5938	0.4582	0.5422
Trial 4	0.4077	0.5923	0.4761	0.5310
Trial 5	0.4064	0.5934	0.4892	0.5290
Average Performance	0.4065	0.5935	0.4695	0.5353
(Standard Deviation)	(0.0155)	(0.0007)	(0.0130)	(0.0068)

* Conventional model: log-model version of the unconstrained spatial interaction model of the gravity type; the trials differ in initial conditions as well as in the random sequence of the input signals; average performance is the mean over the given performance values; the testing set consists of 348 points

The average prediction quality measured in terms of ARV and R^2 and averaged over the five trials is 0.4065 and 0.5935, repectively, compared to 0.4695 (ARV) and 0.5353 (R^2) for the gravity model. The prediction quality is rather stable over the different trials. The gradient descent technique, even with a momentum term included, is not a particularly efficient algorithm for error function minimization. Local minimization algorithms, such as the gradient descent, find local minima efficiently and operate best in unimodal problems. When gradients can vary a lot, the search process may progress too slowly when the gradient is small and may overshoot when the gradient is large.

This example illustrates the attractivity of backpropagation network models as function approximators in a spatial context, with noisy real world telecommunication data of limited record length. In order to improve the performance of the model further we need to be able to reduce the bias while at the same time also reducing the variance. White (1990) has shown how the complexity of a single hidden layer network must grow in relation to the data set size in order to be consistent. But this does not tell us the complexity required for any given number of data points. Much more research is needed to get deeper insights into the complex relationship between learning and generalization or in other words to optimize the complexity of a model in oder to achieve the best generalization.

9. Outlook

Computational neural networks provide not only novel and extremely valuable classes of data-driven mathematical tools as illustrated in this paper, but also an appropriate framework for re-engineering our well established spatial analysis tools to meet the new large scale data processing needs in data rich environments. The most important challenges in the years to come are, first, to develop application domain specific methodologies relevant for spatial analysis; second, to gain deeper theoretical insights into the complex relationship between training and generalization which is crucial for the success of real world applications; and, third, to deliver high performance computing on neurohardware to enable rapid CNN prototyping to take place with the ultimate goal to develop application domain specific automatic CNN-systems. This is crucial for making CNNs just another element in the toolbox of spatial analysts.

References

Birkin M, Clarke M, George F 1995 The use of parallel computers to solve nonlinear spatial optimisation problems: An application to network planning. **Environmental and Planning A** 27(7): 1049-68

Burrough P, Craglia M, Masser I, Salgé F 1997 Geographic information: The European dimension, Position Statement from the European Science Foundation’s GISDATA Programme; URL: http://www.shef.ac.uk/uni/academic/D-H/gis/policy.html

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

Carpenter G A, Grossberg S 1987b ART 2: Stable self-organizing of pattern recognition codes for analog input patterns, **Applied Optics** 26: 4919-30

Carpenter G A, Grossberg S, Reynolds, 1991a ARTMAP supervised rel-time learning and classification of nonstationary data by a self-organizing neural network, **Neural Networks** 4: 565-588

Carpenter G A, Grossberg S, Rosen D B, 1991b Fuzzy ART stable learning and categorization of analog patterns by an adaptive resonance system **Neural Networks** 4: 759-771

Fischer M M 1995 Fundamentals in neurocomputing. In Fischer M M, Sikos T T, Bassa L (eds) **Recent developments in spatial information, modelling and processing**. Budapest, Geomarket Co: 31-41

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

Fischer M M, Gopal S 1994 Artificial neural networks. A new approach to modelling interregional telecommunication flows. **Journal of Regional Science** 34 (4): 503-27

Fischer M M, Gopal S 1993 Neurocomputing - a new paradigm for geographic information processing. **Environment and Planning A** 25: 757-60

Fischer M M, Staufer P, 1997, Optimization in a backpropagation neural network environment: with a performance test on a real world pattern classification. WSG-Discussion Paper 63/97, Department of Economic and Social Geography, WU-Wien [available from the authors]

Fischer M M, Gopal S, Staufer P, Steinnocher K 1994 Evaluation of neural pattern classifiers for a remote sensing application. Paper presented at the 34th European Congress of the Regional Science Association, Groningen, August 1994

Gallant S I, 1995 **Neural Network Learning and Expert Systems**. The MIT Press, Cambridge (MA)

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

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

Grossberg S 1976a Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. **Biological Cybernetics** 23: 121-34

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

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

Kim T J, Wiggins L L, Wright J R (eds) 1990 **Expert Systems: Applications to urban and regional planning**. Kluwer, Dordrecht: 191-201

Kohonen T 1982 Self-organized formation of topologically correct feature maps. **Biological Cybernetics** 43: 59-69

Kohonen T 1988 **Self-Organization and Associative Memory**. Springer Verlag, Heidelberg

Leung Y 1997 Feedforward neural network models for spatial pattern classification. In Fischer M M, Getis A (eds) **Recent developments in spatial analysis**. Springer, Berlin [in press]

Leung Y 1993 Towards the development of an intelligent support system. In Fischer M M, Nijkamp P (eds) **Geographic information systems, spatial modelling, and policy evaluation**. Springer, Berlin: 131-45

Leung Y, Leung K S, Fischer M M, Ng W, Lau M K 1996 The evolution of multilayer feedforward neural networks for spatial interaction using genetic algorithms. Working Paper, Department of Geography, The Chinese University of Hongkong

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

Openshaw S 1993 Modelling spatial interaction using a neural net. In Fischer M M, Nijkamp P (eds) **Geographic information systems, spatial modelling, and policy evaluation**. Springer, Berlin: 147-64

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

Rumelhart D E, Hinton G E, Williams R J 1986 Learning internal representations by error propagations. In Rumelhart D E, McClelland J L (eds): **Parallel Distributed Processing: Explorations in the Microstructures of Cognition**, **Vol. 1**. MIT Press, Cambridge (MA): 318-62

Smith T R, Penquet R, Menon S, Agarwal P 1987 KBGIS-II. A knowledge-based geographical information system. **International Journal of Geographic Information Systems** 1: 149-72

Webster C 1990 Rule-based spatial search. **International Journal of Geographic Information Systems** 4: 241-59

Weigend A S, Rumelhart D E, Huberman B A 1991 Back-propagation, weight-elimination and time series prediction. In Touretzki D S, Elam J L, Sejnowski T J, Hinton G E (eds): **Connectionist Models: Proceedings of the 1996 Summer School**. Morgan Kaufmann Publishers, San Mateo (CA): 105-16

White H 1990 Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. **Neural Networks** 3: 535-50