Computational Neural Networks: An attractive class of mathematical models for transportation research

Fischer, Manfred M.

Published: 01/05/1997

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
WSG 57/97

Computational Neural Networks:
An attractive class of mathematical models
for transportation research

Manfred M. Fischer
Computational Neural Networks:
An attractive class of mathematical models
for transportation research

*Manfred M. Fischer*
1. Introduction

Most of the transportation research 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, and exhibit major difficulties to cope with the information rich worlds on which societies and economies increasingly depend.

No doubt, transportation research is currently entering a period of rapid change, a period which presents the unique opportunity for new 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. It is argued in this chapter that computational intelligence technologies in general and computational neural networks in particular show the potential for a new paradigm in transportation research providing transportation researchers with rich and interesting classes of novel data driven methods and techniques applicable to a wide range of domains in transportation research.

The chapter is structured as follows. First, we attempt to clarify what we mean with computational neural networks [CNN] in contrast to artificial neural networks [ANN] (see section 2). Ignoring the distinction between artificial neural networks and computational neural-like networks leads to confusion, misunderstanding, misrepresentation and misuse of neural network models in transportation research. In section 3 we consider some fundamental characteristics of these computational neural networks. It would be impossible to consider all the different computational neural network models and their associated learning techniques in a single chapter. Here we concentrate on feedforward neural networks which provide transportation researchers with a novel and extremely useful class of mathematical tools. Section 4 deals with supervised training of such networks and briefly reviews some powerful (local) optimization techniques. Section 5 finally provides a small-sized real-world example for interregional telecommunication traffic modelling and illustrates its superiority compared with the standard statistical benchmark. Some conclusions are provided in the final section.

2. What are Computational Neural Networks?

The recent emergence of computational technologies such as artificial life, evolutionary computation, neural networks has been accomplished by a virtual explosion of research, spanning a range of disciplines perhaps wider than any other contemporary intellectual
endeavour. Researchers from such diverse fields such as neuro-science, computer science, cognitive science, physics, engineering, statistics, mathematics, computational geography and economics are daily making substantial contributions to the understanding, development and applications of computational neural like systems that mimic certain aspects of the form and functionality of human intelligence.

Computational intelligence (CI) - and this is important to stress - denotes the lowest-level forms of 'intelligence' which stem from the ability to process numerical (low-level) data without using knowledge in the AI sense (see Bezdek 1994). An artificially intelligent system is a CI system where added value comes from incorporating knowledge of non-numerical information, rules and constraints that humans process. Thus, neural networks such as multi-layer perceptrons, self-organizing maps, learning vector quantization, adaptive resonance theories etc. are generally CI rather than AI systems. Ignoring the distinction between artificial neural networks and computational neural-like networks may lead to confusion, misunderstanding, misrepresentation and misuse of neural network models in transportation research.

Computational neural networks are parallel distributed information processing structures consisting of simple, but generally non-linear processing (computational) elements which can possess a local memory and can carry out localized information processing operations with adaptation capabilities, massively interconnected via unidirectional signal conduction paths called connections. Each connection has a weight associated with it that specifies the strength of this link. A processing element (PE) can receive any number of incoming connections and has a single output connection which can branch into copies to form multiple output connections, where each carrying the same signal. The information processing active within each PE can be defined arbitrarily with the restriction that it has to be completely local, i.e. it has to depend only on the current values of the input signals arriving at the PE and on values stored in the PE's local memory (see Hecht-Nielsen 1990).

Figure 1: A typical neural network architecture
Figure 1 shows a typical neural network architecture. The input to the network considered as a data array $\xi$ and the output of the network as a data array $\zeta$. Viewed in this manner the general functional form of a network is similar to that of a software procedure 'input' 'processing' 'output'. Whether implemented in parallel hardware or simulated on a von Neumann computer, all computational networks consist of a collection of simple computational elements that work together to solve problems.

Figure 2 reflects current thinking about information processing that should be performed at each PE in a computational neural network. Characteristically, two mathematical functions are active at each PE. The first mathematical function is an integrator function, say $f$, which integrates the connection weights, say $w=\{w_i\}$, with the input signals, say $\xi=\{\xi_i\}$, arriving via the incoming connections which impinge upon the processing element. The first entry in each vector in Figure 2 is shown by a dotted line to indicate the bias weight $w_0$ connected to a constant input $\xi_0=1$. Typically $f$ is the inner product, usually the Euclidean dot product, say

$$\eta = f(\xi) = \langle \xi, w \rangle = \sum_{i=1}^{k} \xi_i w_i + w_0$$  \hspace{1cm} (1)

where $w_0$ is an unknown parameter which has to be predefined or learned during training. $w_0$ represents the offset from the origin of $\mathbb{R}^k$ to the hyperplane normal to $w$ defined by $f$. Without loss of generality the augmented vectors $\xi=(1, \xi_1, ..., \xi_k)$ and $w=(w_0, w_1, ..., w_k)^T$ may be considered as input and weight vectors, respectively, in $\mathbb{R}^{k+1}$. A processing element with this type of integrator function is called first-order processing element because $f$ is an affine function of its input vector $\xi$. When the inner product $f$ is replaced by a more complicated function, higher-order processing elements arise. For example, a second order processing element may be realised with a quadratic form, say $\xi^T w \xi$, in $\xi$. It is important to note that each processing element may be viewed as having (k+1) unknowns, but only k inputs.

Each processing element typically applies a transfer (or activation) function, say $F$, to the value of the integrator function (or activation) on its inputs. The transfer function produces the processing element’s output signals. The transfer functions in many fundamental CNNs satisfy

$$F(\eta) = \begin{cases} 1 & \text{as } \xi^T \eta + 8 \\ 0 & \text{as } \xi^T \eta - 8 \end{cases}$$  \hspace{1cm} (2)

which are called sigmoid functions. A common value choice in the case of continuous inputs is the logistic function shown in figure 2.

A third mathematical operation for a computational neural network is the update function
\( W(t+1) = U(W(t)) \) \hspace{1cm} (3)

where \( W(t) = (w_1(t), ..., w_N(t)) \) denotes the network weight vector, i.e. the collection of all the individual vectors at the \( N \) PEs in the network at any time (iteration) \( t \). The weight vector \( w_n(n=1,...,N) \) is stored in the \( n \)-th processing element's local memory. Updating is done during training.

**Figure 2: Information processing at the processing element**

![Diagram of Information processing at the processing element]

Although a vast variety of NN models exist, and more continue to appear as research continues, many of them have common topological characteristics, properties of the PEs, and training [learning] approaches (see, e.g., Hecht-Nielsen 1990, Carpenter and Grossberg 1991, Kosko 1992, Wasserman 1993). Basically three entities characterize a computational NN (see, for example, Fischer and Gopal 1993):

- the network topology or interconnection of its PEs [called *architecture*],
- the characteristics of its PEs, and
- the method of determining the weights at the connections [called *training or learning strategy*].
Different interconnection strategies lead to different types of NN architectures (for example, feedforward versus recurrent) which require different learning (training) strategies. At the most fundamental level two categories of training may be distinguished: supervised and unsupervised. In supervised learning the network is trained on a training set consisting of a sequence of input and target output data. Training is accomplished by adjusting the network weights so as to minimize the difference between the desired and actual network outputs. Unsupervised learning (also called self-organization) requires only input data to train the network. During the training process the network weights are adjusted so that similar inputs produce similar outputs. This is accomplished by a training algorithm that extracts statistical regularities from the training set, representing them as the values of network weights (see Fischer and Gopal 1994b, Fischer 1995).

3. The Class of Feedforward CNNs

Multilayer feedforward computational neural networks such as perceptrons and radial basis function networks have recently emerged as attractive class of CNNs based upon sound theoretical concepts. They map $\xi_1, \ldots, \xi_I$ inputs into $\zeta_1, \ldots, \zeta_P$ outputs, and may be viewed as generalized non-linear extensions of conventional spatial statistical models such as, e.g., spatial regression models, spatial interaction models, and linear discriminant functions. To better understand this relationship, we explicitly express feedforward CNNs mathematically, and consider for this purpose feedforward CNNs with inputs $\xi_1, \ldots, \xi_I$, one hidden layer of $j=1, \ldots, J$ computational units and - for simplicity's sake - one output unit $\zeta_p$ only. Such networks may be mathematically expressed as

$$
\zeta = g_2 \left( \sum_{j=0}^{J} w_j^{(2)} \left( \sum_{i=0}^{I} w_{ij}^{(1)} \xi_i \right) \right) = g(\zeta, \theta) \tag{4}
$$

where the parameters $w_{ij}^{(1)}$ (i=1,..,I; j=1,..,J) denote weights associated with connections from the input array of I units to the hidden layer, and the parameters $w_j^{(2)}$ (j=1,..,J) those weights associated with connections from the hidden layer to the output unit. The bias have been absorbed into the weights. $g_1$ and $g_2$ represent transfer functions of the PEs at the hidden and output layer, respectively. The expression $g(\zeta, \theta)$ is a convenient short-hand for network output since this depends only on inputs and weights. The symbol $\xi$ represents a vector (list) of all the input values, and the symbol $\theta$ represents a vector of all the weights (the $w_{ij}^{(1)}$s and $w_j^{(2)}$s). $g$ might be called the network output function. The transfer functions of the hidden and output unit determine the precise form of the function $g$.

Different types of transfer functions $g_1$ and $g_2$ will lead to different particular computational networks. If the transfer functions are taken to be linear so that $g_1(a)=g_2(a)=a$, functional form
The general linear regression model. The crucial difference is that here we consider the weight parameters appearing in the hidden and output layers as being adaptive so that their values can be changed during the process of network training (in statistical terminology: parameter estimation).

The novelty and fundamental contribution of the feedforward neural network approach to transportation research derives from its focus on functions such as $g$ in (4), and much less on the associated learning methods which will be discussed in section 4. Among others Hornik et al. (1989) have demonstrated that the network output function $g$ can provide an accurate approximation to any function of $\xi$ likely to be encountered, provided that the number $J$ of hidden units is sufficiently large. Because of this universal approximation property, one hidden layer feedforward networks are useful for applications in travel demand analysis in general and traffic flow modelling in particular, and a host of related tasks.

Feedforward CNN modelling as universal function approximators may be considered as a three-stage process as outlined in Fischer and Gopal (1994a):

1. The first stage refers to the identification of a model candidate from a family of two-layer feedforward networks with specific types of non-linear processing elements.

2. The second stage involves the estimation of the network parameters of the selected neural network model and the optimization of the model complexity for the given training set.

3. The third stage is concerned with testing and evaluating the out-of-sample [generalization] performance of the model.

One critical issue for a successful application of CNNs to transportation research is the complex relationship between learning (training) and generalization. It is important to stress that the ultimate goal of network training is not to learn an exact representation of the training data itself, but rather to build a model of the process which generates the data in order to achieve a good generalization [out-of-sample] performance of the model. One way to optimizing the generalization performance of a model is to control its effective complexity where complexity being measured in terms of network parameters. This problem of finding the optimal complexity for a neural network model - though crucial for a successful application - has been highly neglected in applications up to now. In principle, there are three classes of techniques to control overfitting of a model:

- **Regularization techniques** [i.e. add a penalty term to the error function],

- **Network pruning techniques** [i.e. train an overly-large network and successively delete weights, as illustrated, e.g., in Fischer et al. 1994], and
**cross-validation techniques** to determine when to stop training [i.e. early stopping heuristic, as illustrated, e.g., in Fischer and Gopal 1994a].

The point of best generalization is determined by the trade-off between the bias and the variance of the model, and occurs when the combination of bias and variance is minimized. In the case of feedforward networks it is possible - by using a sequence of successively larger data sets, and a corresponding set of models with successively greater complexity - to reduce both bias and variance simultaneously and, thus, to improve the generalization performance of the model. The generalization performance which might be achieved is, however, limited by the intrinsic noise of the data (see Fischer 1996).

**4. Training of Feedforward CNNs**

Given a sufficiently complex neural network model [i.e., J sufficiently large], the essence of network learning is to find a suitable set of parameters that approximate an unknown input-output relation of type (4). This problem is generally solved using supervised learning techniques. Supervised learning requires a training set (i.e., a set of input-target output examples). Learning the training set may be posed as a search in the network parameter space by introducing a performance (error) measure, i.e. a function of the adaptive network parameters, that measures the quality of the network's approximation to the input-output relation on the restricted domain covered by the training set. The minimization of this error over the network's parameter space is called the training process. The task of learning, however, is to minimize that error for all possible examples related through the input-output relation, namely, to generalize outside of the training set.

A frequently encountered performance measure is the least mean squared error (for more details see, e.g., Gopal and Fischer 1996a)

\[ E ( \zeta^* \mid \xi, \theta) = \frac{1}{2} ( \zeta^* - g (\xi, \theta))^2 \] (5)

for a training set available consisting of a vector (list) \( \xi \) of all the input training patterns together with observations on corresponding target variables \( \zeta^* \).

CNN training strives to minimize the chosen error function such as (5). Due to the non-linearity of the transfer functions \( g_1 \) and \( g_2 \), it is not possible to find closed-form solutions for this optimization problem. But there is a considerable variety of local and global search procedures available (see figure 3). Minimization techniques are termed local if the computations needed to update each network weight (see equation (3)) can be performed using...
local information to that weight only. This may be motivated by the desire to implement network training algorithms in parallel hardware.

Local training techniques generally involve an iterative procedure to minimize a performance measure such as (5), with adjustments to the network parameters $\theta$ being made in a sequence of iteration steps. At each iteration step we can distinguish between two different stages. In the first stage, the derivatives of the performance function with respect to the network parameters have to be evaluated. This evaluation is most commonly performed by the backpropagation technique which provides a computationally efficient procedure for evaluating such derivatives (Rumelhart et al. 1986). At this stage errors are propagated backwards through the network to the output processing units. It is important to note that the backpropagation technique can also be used for the evaluation of other derivatives such as the Jacobian and Hessian matrices. In the second stage, the derivatives are utilized to compute the parameters adjustments. For this stage of weight adjustment a wide range of optimization procedures may be used (Bishop 1995).

Figure 3: A taxonomy of CNN training procedures

The simplest and most popular of such optimization procedures is the gradient (also known as steepest) descent. Gradient descent techniques involve taking a sequence of iteration steps through the parameter space. With the simple gradient descent the direction of each step is given by the local negative gradient of the performance function chosen, while the step size performed is determined by an arbitrary parameter, called learning rate.

In the pattern-based (also termed on-line) version of the gradient descent, the error function gradient is evaluated for just one training pattern at a time and the parameter values updated where the different patterns in the training set may be used either in sequence (deterministic pattern-based version) or selected at random (stochastic pattern-based version). The stochastic version shows the potential advantage to escape from local minima. The pattern-based versions
tend to be superior to the batch version especially for large and redundant training sets (Hertz et al. 1991). The training of CNNs using the backpropagation technique in combination with the basic gradient descent is plagued by slow convergence in the case of larger training sets. Numerous heuristic optimization algorithms have been proposed to improve the convergence speed of the gradient descent technique. Examples include the gradient descent with momentum update, the delta-delta rule, the delta-bar-delta rule (see Jacobs 1988) and a heuristic scheme known as quickprop (Fahlman 1988), to mention just the most popular ones.

Another important class for weight adjustment is based on the concept of conjugate gradients. Conjugate gradient procedures provide minimization techniques which require only the evaluation of the error function and its derivation, and utilize information about the direction search from the previous iteration in order to accelerate the convergence. Each search direction is conjugate if the performance function is quadratic. Theoretically, this procedure guarantees to minimize a quadratic error function in $q$ or fewer iterations (batch mode) where $q$ is the dimensionality of the parameter space. It is interesting to note that the conjugate gradient procedure may be regarded as a form of gradient descent with momentum, where the learning rate is determined by line search.

Quasi-Newton (also called variable-metric) procedures, the third class of local search techniques, are today the most efficient and sophisticated optimization algorithms, including the Davidson-Fletcher-Powell (DFP) and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) techniques. They iteratively use successively improved approximations to the inverse Hessian instead of the true inverse as in the basic Newton procedure, and utilize only information on the first derivatives of the performance function. Quasi-Newton procedures require much more storage, but only half of the gradient evaluations in comparison to conjugate gradient descents. In cases of CNNs with more than a few thousand network parameters procedures such as conjugate gradients have significant advantage over the quasi-Newton techniques (Shanno 1990).

Local minimization algorithms find the local minima efficiently and work best in unimodal problems. They show difficulties when the surface is flat (i.e. gradients close to zero), when gradients can be in a large range, or when the surface is very rugged. To overcome local search deficiencies, global minimization procedures such as simulated annealing and genetic search algorithms may be used.

5. An Example: Interregional Telecommunication Traffic Modelling

This section serves to demonstrate the feasibility of the single hidden layer feedforward neural network approach with logistic functions $g_1$ and $g_2$ to model interregional telecommunication traffic in Austria and to evaluate its performance in comparison with the classical approach of
the gravity type. This approach belongs to the class of (unconstrained) spatial interaction models of the gravity type and generally assumes (see Rietveld and Janssen 1990, Fischer et al. 1992) that

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

(6)

with

$$F_{rs}(D_{rs}) = D_{rs}^{\alpha_3} \quad r, s = 1, ..., n$$

(7)

where $T_{\text{conv}}^{rs}$ denotes the intensity of telecommunication from region $r$ to region $s$ ($r, s = 1, ..., n$) [measured in terms of erlang or minutes] predicted by (6)-(7). $A_r$ represents the intensity of telecommunication traffic generated by region $r$, and $B_s$ destination-specific pull factors. In this study we have decided to use gross regional product as a measure for $A_r$ and $B_s$. Gross regional product as a proxy of economic activity and income is relevant for both business and private telecommunication (see Rietveld and Janssen 1990). $F_{rs}$ is a factor associated with origin-destination pairs ($r, s$) representing the inhibiting effect of geographic separation, specified in form of a power function in $D_{rs}$ [see equation (7)]. $D_{rs}$ denotes the distance from region $r$ to region $s$. $K$ is a scale parameter (constant), $\alpha_1$, $\alpha_2$ and $\alpha_3$ are parameters to be estimated.

The usual strategy to estimating (6)-(7) is to assume that a normally distributed error term $\varepsilon_{rs}$ applies, i.e. $\varepsilon_{rs} \sim N(0, \sigma)$ independently of and $A_r$, $B_s$ and $D_{rs}$. OLS estimation can be applied then after a logarithmic transformation of (6)-(7), leading to unbiased and consistent estimates $\hat{\alpha}_1$, $\hat{\alpha}_2$ and $\hat{\alpha}_3$.

This log-normal model version serves as benchmark to evaluate the relative efficiency of the neural net model derived from the general class of feedforward CNNs by the following specifications:

- $p_I = 3$ input units corresponding to the three independent variables $A_r$, $B_s$ and $D_{rs}$ in (6),
- $p_O$ one output unit $\zeta$ corresponding to the dependent variable $T_{\text{neur}}^{rs}$, the predicted telecommunication intensity from $r$ to $s$, and
- logistic transfer functions $g_1$ and $g_2$.

Then the neural net model reads as follows

$$\zeta = \left[ 1 + \exp \left( - \left( \sum_{j=0}^{J} w_{(2)}^{(2)} \left( 1 + \exp \left( - \left( \sum_{i=0}^{3} w_{(1)}^{(1)} \xi_{(1)}^{(1)} \right) \right) \right) \right) \right) \right]^{-1} = g(\xi, \theta)$$

(8)
where the bias has been absorbed into the weights for convenience. This model may be viewed as a non-linear regression function with a quite specific form.

The problem of finding a suitable set of $\theta$-parameters which approximate the unknown input-output relation (8) is tackled by means of supervised training. The telecommunication data used stem from network measurements of carried traffic in Austria in 1991, in terms of erlang, an internationally widely used measure of telecommunication contact intensity, which is defined as the number of phone calls including facsimile transfers multiplied by the average length of transfer divided by the duration of measurement. The data refer to the total telecommunication traffic between the thirty-two telecommunication districts [i.e. $n=32$] representing the second level of the hierarchical structure of the Austrian telecommunication network (see figure 3). Due to the measurement problems, intraregional traffic (i.e. $r=s$) is left out of consideration here. Thus, the data for the model building process is made up of 992 4-tuples $(A_r, B_s, D_{rs}; T_{obs})$ with $r, s = 1, ..., 32$ and $r \neq s$. The first three components represent the input and $T_{obs}$ the desired output [see Fischer and Gopal 1994a for more details]. In order to make the comparisons with the log-normal version of the benchmark model (6)-(7) as closely as possible, the input and output signals were preprocessed to logarithmically transformed data scaled to (0,1).

Estimation of $\theta$ has been performed by minimizing the least mean square error performance measure (5) and applying the error backpropagation technique in epoch rather than batch mode [epoch size of 20 input signals presented in random order; i.e., stochastic approximation]. Five different random initializations [drawn at random from a uniform distribution between -0.1 and 0.1] were used to capture different parameter variations due to different random initial conditions. Simulations under various parameter settings showed that it was best to use a small learning rate of 0.15 and a momentum of 0.8 across all the trials.

It is important to note that the overfitting problem [see section 3] which destroys the generalization capability (out-of-sample performance) of the model is particularly serious for noisy real world data of limited record length. If the network mapping [i.e. $g$] fits the training data exactly, the capability of the CNN to generalize, that is to generate an acceptable output for novel inputs will often be rather poor. This arises from the rapid oscillations in the network function that are generally needed to fit the noisy training data. To improve the generalization performance of (8) it is necessary for $g$ to represent the underlying trends in the data, rather than fitting all of the fine details of the data.
There are several strategies for handling the overfitting problem [see section 3]. One appealing method is the random method of cross-validation to detect when overfitting occurs. In this case it is crucial to split the whole available data set of 992 patterns into three separate subsets: a training [estimation] set, a validation test set and a testing [prediction] data set. While the training set is directly used for training CNN defined by equation (8), the validation test set is used only for the evaluation process, i.e. for determining when to stop the training process. The testing or prediction set is strictly set apart and never used in the estimation stage. The training set-validation set pair used in this study consists of two thirds of the available data. One quarter of these data (i.e. 148 patterns) have been used for the validation process. It is crucial that the validation test set is only utilized to detect a statistically proper stopping point of the training process of the CNN model identified in the first step of the model building process [see section 3]. Model (8) with J=30 [i.e. 30 hidden processing elements] was identified as most appropriate for the problem at hand.

Figure 4 shows the performance of this CNN [i.e. J=30] as a function of training time in epochs with an epoch size of 20 patterns. The average relative variances are given for the training set, the validation set and the testing (prediction) set. This ARV-performance measure is defined as

$$\text{ARV}(S) = \frac{\sum_{i \in S} (\zeta_i^* - \zeta_i)^2}{\sum_{i \in S} (\zeta_i^* - \bar{\zeta})^2} = \frac{1}{N_s} \frac{1}{N_s} \sum_{i \in S} (\zeta_i^* - \zeta_i)^2$$

(9)

where $\zeta_i^*$ denotes the target value [i.e. the observed $T_{rs}$] and $\zeta_i$ the actual network value, $\bar{\zeta}$ the average over the 20 desired values in S. The averaging, i.e. division by $N_s$ [the number of patterns in set S (=epochs), $N_s=20$] makes ARV(S) independent of the size of the set. The
division by the estimated $\hat{\sigma}^2$ of the data removes the dependence on the dynamic range of the data. This implies that if the estimated mean of the observed data would be taken as predictor, ARV(S) would be equal to 1 (see Weigend et al. 1991). In this study the variances of the individual sets S associated with the different epochs differ only slightly. Thus, it appears to be reasonable to always use the variance of the entire data record $\hat{\sigma}^2 = \sigma^2_{\text{all}} = 3.112$ as a proxy in order to facilitate computations.

Figure 4 illustrates that the ARV-error of the CNN model measured using the training set continuously decreases and seems to level out after 5,000 epochs. This is what one expects for a CNN. The validation test set error as shown in figure 4 decreases first, after 1,500 epochs only at a moderate rate until 4,250 epochs, then slightly increases and tends to approach an asymptotic value. If we assume that the error curve of the 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 crudely correct 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 4, 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.

Figure 5: Training, validation and prediction set curves of the CNN model with J=30 as a function of training time in epochs (the vertical line P indicates the stopping point)

The ultimate goal of the modelling exercise is to predict interregional telecommunication traffic flows. Thus, we briefly discuss the predictive quality of the CNN model (8) and compare it with
that of the classical model (6)-(7). To assess the prediction performance, we primarily use the average relative variance ARV(S) as defined in (9) and the coefficient of determination $R^2(S)$ defined in an analogous manner.

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

<table>
<thead>
<tr>
<th>Prediction (Testing)</th>
<th>Neural Net Model</th>
<th>Classical Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARV</td>
<td>$R^2$</td>
</tr>
<tr>
<td>Trial 1</td>
<td>0.4063</td>
<td>0.5937</td>
</tr>
<tr>
<td>Trial 2</td>
<td>0.4057</td>
<td>0.5942</td>
</tr>
<tr>
<td>Trial 3</td>
<td>0.4063</td>
<td>0.5938</td>
</tr>
<tr>
<td>Trial 4</td>
<td>0.4077</td>
<td>0.5923</td>
</tr>
<tr>
<td>Trial 5</td>
<td>0.4064</td>
<td>0.5934</td>
</tr>
<tr>
<td>Average Performance</td>
<td>0.4065</td>
<td>0.5935</td>
</tr>
<tr>
<td>(Standard Deviation)</td>
<td>(0.0155)</td>
<td>(0.0007)</td>
</tr>
</tbody>
</table>

* 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

Table 1 reports the prediction performance of the CNN model on the testing set in terms of ARV and $R^2$ averaged over the five trials. In order to make a fair comparison, the classical model was estimated (tested) with the same data seen by the neural net model during training (testing). The CNN model evidently shows a somewhat higher prediction performance in all trials. The average prediction quality in terms of ARV and $R^2$ is 0.4065 and 0.5932 respectively, compared to 0.4695 (ARV) and 0.5353 ($R^2$) for the classical model. The prediction quality is rather stable over the different trials. The prediction accuracy achieved by the two alternative interregional teletraffic models is exemplified by a systematic sample of $T_{rs}$-values in table 2.

Table 2: Prediction accuracy of the computational neural network and the conventional statistical benchmark model: Some selected results

<table>
<thead>
<tr>
<th>$T_{rs}$</th>
<th>Observation</th>
<th>Model Predictions</th>
<th>Conventional</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neural Net</td>
<td>Conventional</td>
</tr>
<tr>
<td>$T_{2,17}$</td>
<td>69.7279</td>
<td>44.2051</td>
<td>57.8402</td>
</tr>
<tr>
<td>$T_{3,30}$</td>
<td>0.2113</td>
<td>0.1219</td>
<td>0.7010</td>
</tr>
<tr>
<td>$T_{4,7}$</td>
<td>12.1801</td>
<td>11.5071</td>
<td>13.8977</td>
</tr>
<tr>
<td>$T_{5,21}$</td>
<td>1.7500</td>
<td>1.6858</td>
<td>2.1387</td>
</tr>
<tr>
<td>$T_{11,23}$</td>
<td>0.4314</td>
<td>0.4695</td>
<td>0.5587</td>
</tr>
<tr>
<td>$T_{13,7}$</td>
<td>1.4137</td>
<td>1.8286</td>
<td>2.3708</td>
</tr>
<tr>
<td>$T_{16,9}$</td>
<td>10.4940</td>
<td>10.8594</td>
<td>19.4737</td>
</tr>
<tr>
<td>$T_{18,4}$</td>
<td>21.9815</td>
<td>20.5638</td>
<td>23.3007</td>
</tr>
<tr>
<td>$T_{20,11}$</td>
<td>4.4647</td>
<td>3.7864</td>
<td>3.4129</td>
</tr>
<tr>
<td>$T_{28,16}$</td>
<td>1.5310</td>
<td>1.6492</td>
<td>1.6535</td>
</tr>
<tr>
<td>$T_{29,18}$</td>
<td>15.082</td>
<td>19.7773</td>
<td>30.9833</td>
</tr>
</tbody>
</table>
The analysis unequivocally shows that the neural net model outperforms the classical statistical benchmark model in terms of both the ARV and \( R^2 \) prediction performance as well as the prediction accuracy indicated by a root-mean square error of 20.52 percent compared to 24.56 percent for the benchmark. In some cases, the telecommunication flows are replicated substantially more accurately. The primary advantage of the computational neural network model over the classical statistical model lies in the fact that it has a more general non-linear functional form than the conventional model can effectively deal with. This functional form can be viewed as a non-linear regression function of a quite specific form (White 1989), and thus as a generalization of the linear design of the benchmark model.

6. Conclusions and Outlook

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

Finally, it is important to stress that by casting the analysis in the relatively universal language of mathematics, makes it possible to dispel much unwarranted mystique perceived by those outside the field of computational neural networks which arises from the origins of computational neural systems in the study of natural neural systems, and in the associated metaphorical jargon in the field. This provides a firm foundation on which to base a study of the capabilities and limitations of such models and may act to diminish the misuse and misunderstanding of the paradigm of computational neural networks which is currently rather widely spread over many disciplines.

References


Jacobs R A 1988 Increased rates of convergence through learning rate adaptation, Neural Networks 1, pp. 295-307.


