Pergamon

Neural Networks, Vol. 9, No. 5, pp. 855–869, 1996

Printed in Great Britain
0893–6080/96 $15.00+.00

0893–6080(95)00116–6

CONTRIBUTED ARTICLE

N-tuple Regression Network

ALEKSANDER KOLCZ AND NIGEL M. ALLINSON

University of York

(*Received* 14 *July* 1994; *revised and accepted* 19 *August* 1995)

Abstract—*N-tuple neural networks (NTNNs) have been successfully applied to both pattern recognition and function approximation tasks. Their main advantages include a single layer structure, capability of realizing highly non-linear mappings and simplicity of operation. In this work a modification of the basic network architecture is presented, which allows it to operate as a non-parametric kernel regression estimator. This type of network is inherently capable of approximating complex probability density functions (pdfs) and, in the limiting sense, deterministic arbitrary function mappings. At the same time, the regression network features a powerful one-pass training procedure and its learning is statistically consistent. The major advantage of utilizing the N-tuple architecture as a regression estimator is the fact that in this realization the training set points are stored by the network implicitly, rather than explicitly, and thus the operation speed remains constant and independent of the training set size. Therefore, the network performance can be guaranteed in practical implementations.*

Keywords—Non-parametric estimation, Regression equation, *N*-tuple sampling, RAM-based neural networks, Local receptive fields, Basis functions.

1. INTRODUCTION

Establishing an approximate input/output relationship for a multi-dimensional, generally non-linear system is one of the major application areas of artificial neural networks. Potential uses include system identification and prediction characteristic of control problems. Many of the existing network architectures, including multilayer perceptrons (MLPs), radial basis functions (RBFs), cerebellar model articulation controller (CMAC) and NTNN, have been applied for this purpose. Indeed, the recent theoretical results concerning universal function approximation properties guarantee that certain architectures (i.e., MLPs and RBFs) are potentially capable of achieving the approximation to any desired level of accuracy, given enough memory storage and processing power of their implementations (Hornik et al., 1989; Park & Sandberg, 1991).

The recently investigated general regression neural network (GRNN) (Specht, 1991) approximates an unknown system mapping by estimating the expected value of the system output given a specific value of the input (i.e., by estimating the regression function) and a finite set of training samples, where the system input and output are vectors of random variables. The network obtains non-parametric estimation of the system probability density function on the basis of a finite training/design set drawn according to this distribution. The GRNN has been shown to perform very well even in the cases of very few training samples (Specht, 1991). It is especially suitable for situations where the available data are inaccurate or corrupted by noise, and offers the advantage of a very simple one-pass learning procedure. One potential implementation-oriented problem of this architecture arises when the whole training set is stored and used during the network operation, which can prove too memory/performance expensive in a practical realization. In such cases data-size reduction procedures (e.g., clustering) may have to be applied as a preprocessing stage.

A particular implementation of the GRNN using a NTNN architecture is proposed, where the training data set size problem can be avoided. In particular, both the network structure and the operation speed

Acknowledgements: The authors are grateful for the generous support offered by the Overseas Research Student Awards Scheme and the University of York in providing a research studentship (UK).

Requests for reprints should be sent to Aleksander Kolcz, Department of Electronics, Image Engineering Laboratory, University of York, Heslington, York YO1 5DD, UK; e-mail: ark@ohm.york.ac.uk

remain constant and independent of the amount of training the network receives. The N-tuple architecture, introduced by Bledsoe and Browning (1959), has found many applications in pattern recognition, image processing and function approximation (Aleksander & Stonham, 1979; Aleksander et al., 1984; Tattersall et al., 1991; Kolcz & Allinson, 1994, 1995). Its operation relies on transforming an arbitrary system input into a binary array and subsequently sampling it, taking N random array locations (i.e., a random N-tuple) at a time. The resulting binary N-element vectors serve as addresses to a set of memory nodes, the elements of which are combined to yield the network response. Many variations of the NTNN differ in the format of the memory locations (e.g., single vs multiple words per location, binary, integer or floating point memory word formats) as well as in the update rules used during learning. In this paper we are concerned mainly with the function approximation variant of the network, also known as the single-layer look-up perceptron (SLLUP) (Tattersall et al., 1991).

Section 2 describes the GRNN and the conditions imposed on architectures for its implementation. Section 3 discusses the mapping inherent in the approximation-type NTNN. Section 4 introduces the modifications which allow the N-tuple architecture to realize a kernel regression estimator, and a derivation of the network mapping is provided. Section 5 presents some architecture-specific issues involved in the implementation of GRNN by NTNN. In Section 6 several simulation results are given, and the paper is concluded in Section 7.

2. ESTIMATION OF THE GENERAL REGRESSION EQUATION

We consider a general system taking a D-dimensional real-valued vector, $\mathbf{x}$, as its input and producing a scalar real-valued output, y (a scalar rather than vector output form is considered for simplicity). The input and output are realizations of random variables $\mathbf{X}$ and Y, respectively. It is assumed that $\mathbf{X}$ and Y are distributed according to a continuous joint probability density function (pdf) $f(\mathbf{x}, y)$. We seek to find an input/output relationship of the system in terms of the regression or a conditional mean of the dependent variable Y for any particular value of the input, $\mathbf{x}$

$$m(\mathbf{x}) = E\{Y/\mathbf{x}\} = E\{Y/\mathbf{X} = \mathbf{x}\} : \mathscr{R}^D \to \mathscr{R} \qquad (1)$$

where it is assumed that the conditional mean exists and is well-defined over the input domain (i.e., $\forall \mathbf{x}\ E|m(\mathbf{x})| < \infty$). For a known underlying pdf the regression function is given by

$$m(\mathbf{x}) = E\{Y/\mathbf{x}\} = \frac{\int_{-\infty}^{\infty} y \cdot f(\mathbf{x}, y)dy}{\int_{-\infty}^{\infty} f(\mathbf{x}, y)dy} = \int_{-\infty}^{\infty} y \cdot f(y/\mathbf{x})dy \qquad (2)$$

and for any particular $(\mathbf{x}, y)$ pair generated by the system

$$y = m(\mathbf{x}) + \varepsilon \qquad (3)$$

where the random error component, ε, disappears in the average (i.e., $E\{Y/\mathbf{x}\} = m(\mathbf{x})$). However, when no explicit knowledge about the system is available, the regression function can only be estimated from a finite set of T random points $(\mathbf{x}^i, y^i)$ taken from the system according to its distribution. Regression analysis plays a major role in statistics and various approaches to the estimation problem exist (Härdle, 1990). In this work we are concerned only with one kind of non-parametric regression, based on the well-established kernel method (Hand, 1982) for probability density estimation. From the definition of the conditional mean (2) it is apparent that if an estimate of the system joint pdf was available, it could be used directly for estimating the regression function. The kernel method provides a means of estimating $f(\mathbf{x}, y)$ with no assumptions being made about its form, allowing approximation of the regression function in the general case.

The kernel estimation of density functions was first investigated by Rosenblatt (1956) and Parzen (1962) for the univariate case, and further extended to multivariate distributions (Cacoullos, 1966). The following discussion draws extensively from the recent monographs of Hand (1982) and Scott (1992). The method relies on assigning a smooth monotonically decreasing function (i.e., the kernel function) to every sample $(\mathbf{x}, y)$ taken from the distribution. The kernels are usually normalized so that they are also valid pdfs. Thus the estimate $\hat{f}(\mathbf{x}, y)$ of $f(\mathbf{x}, y)$ is built as a superposition of all kernel functions associated with the sample points. As this method provides a smooth estimate of the unknown density, it is required that the underlying pdf is also reasonably smooth so as to yield valid results. Thus $f(\mathbf{x}, y)$ is assumed to be continuous and differentiable. Many theorems about consistency and the rates of convergence of the estimate actually require the existence of the second and third order derivatives (Hand, 1982). The estimates provided by the kernel method are generally biased for a finite size of the training set, with the bias disappearing asymptotically provided that certain assumptions about the kernel functions are made.

The univariate, real and even kernel function, $\varphi(\mathbf{x})$, satisfies the following conditions (Parzen, 1962)

$$\sup_{x\in\mathcal{R}}|\varphi(x)| < \infty, \int_{\mathcal{R}}|\varphi(x)|dx < \infty, \lim_{|x|\to\infty}|x\cdot\varphi(x)| = 0. \tag{4}$$

Additionally, if h_T denotes a smoothing parameter (also called the bandwidth or window width of the kernel function) dependent on the number of training samples, T, and satisfying the conditions

$$\lim_{T\to\infty} h_T = 0 \text{ and } \lim_{T\to\infty} T\cdot h_T = \infty \tag{5}$$

then the estimator, $\hat{f}(x)$, given by

$$\hat{f}(x) = \frac{\sum_{i=1}^{T}\varphi((x-x_i)/h_T)}{T\cdot\int_{-\infty}^{\infty}\varphi(x)dx} \tag{6}$$

approaches asymptotically the univariate distribution density $f(x)$. This provides a consistent (in the mean square sense) and asymptotically unbiased estimate of $f(x)$. If

$$\varphi(x)\geq 0 \text{ and } \int_{\mathcal{R}}|\varphi(x)|dx = 1 \tag{7}$$

then $\varphi(x)$ is itself a valid density function. For multivariate kernels similar conditions could be specified.

It is convenient to choose a $(D+1)$-variate kernel function, $\Phi(\mathbf{x},y)$, such that it is separable with respect to $\mathbf{x}$ and y variables, i.e.,

$$\Phi(\mathbf{x},y) = \Phi_x(\mathbf{x})\cdot\varphi_y(y)$$

where $\varphi_y(y)$ is a univariate kernel function satisfying the conditons (4) and (7). With such a kernel the estimate of the density function becomes (assuming that the kernels are normalized to give unit integrals over their respective domains)

$$\hat{f}(\mathbf{x},y) = \frac{1}{T}\sum_{i=1}^{T}\Phi_x(\mathbf{x}-\mathbf{x}^i)\cdot\varphi_y(y-y^i)$$
$$\int_{\mathcal{R}^D}\Phi_x(\mathbf{x})d\mathbf{x} = 1 \text{ and } \int_{-\infty}^{\infty}\varphi_y(y)dy = 1 \tag{8}$$

and the regression function can be estimated as

$$\hat{E}(Y/\mathbf{x}) = \frac{\sum_{i=1}^{T}\Phi_x(\mathbf{x}-\mathbf{x}^i)\cdot\int_{-\infty}^{\infty}y\cdot\varphi_y(y-y^i)dy}{\sum_{i=1}^{T}\Phi_x(\mathbf{x}-\mathbf{x}^i)} = \frac{\sum_{i=1}^{T}y^i\cdot\Phi_x(\mathbf{x}-\mathbf{x}^i)}{\sum_{i=1}^{T}\Phi_x(\mathbf{x}-\mathbf{x}^i)} \tag{9}$$

where $\Phi_x(\mathbf{x})$ is assumed to satisfy the following conditions (analogous to those for the univariate case)

$$\sup_{\mathbf{x}\in\mathcal{R}^D}|\Phi(\mathbf{x})| < \infty, \int_{\mathcal{R}^D}|\Phi(\mathbf{x})|d\mathbf{x} < \infty,$$
$$\lim_{\|\mathbf{x}\|\to\infty}\|\mathbf{x}\|\cdot|\Phi(\mathbf{x})| = 0. \tag{10}$$

This separable form is also called the Nadaraya–Watson kernel regression estimator (Nadaraya, 1964; Watson, 1964). It is often chosen so that the function $\Phi_x(\mathbf{x})$ has a product form as well, which is most commonly used in practical situations. Let $\varphi_d(x_d)$ represent the kernel component of $\Phi_x(\mathbf{x})$ for the dth dimension, and let h_T^d be its smoothing parameter. Then the estimate of the conditional mean with a product kernel is given by

$$\hat{E}(Y/\mathbf{x}) = \frac{\sum_{i=1}^{T}y^i\cdot\Phi_x(\mathbf{x}-\mathbf{x}^i)}{\sum_{i=1}^{T}\Phi_x(\mathbf{x}-\mathbf{x}^i)} = \frac{\sum_{i=1}^{T}y^i\cdot\prod_{d=1}^{D}\varphi_d((x_d-x_d^i)/h_T^d)}{\sum_{i=1}^{T}\prod_{d=1}^{D}\varphi_d((x_d-x_d^i)/h_T^d)}. \tag{11}$$

Asymptotic quadratic consistency of the estimate results if the kernel smoothing parameters satisfy

$$\lim_{T\to\infty} h_T^d = 0 (d = 1,\ldots,D) \text{ and } \lim_{T\to\infty} T\cdot\prod_{d=1}^{D}h_T^d = \infty. \tag{12}$$

A further simplification of the estimate is possible if a common smoothing parameter, h_T, and a common functional form are chosen for all univariate kernel functions. In such cases, although there is a loss of flexibility in optimizing the kernel shape, only one parameter has to be estimated from the training set. It should be noted that assuming a product form for the kernel function does not impose a similar form on

the pdf being estimated. A wide range of functions provide valid kernels, including the commonly used Gaussian function, and the performance tends to be fairly insensitive to the particular kernel used.

Consequently, any network whose final response can be expressed in the form (9), with the function $\Phi_x(\mathbf{x})$ satisfying the stated conditions, naturally implements the non-parametric estimator of the joint pdf and provides an approximate solution to the regression equation. The GRNN proposed by Specht utilizes a product kernel with a common Gaussian univariate kernel form and a common smoothing parameter. However, other variants of the kernel regression estimation have been considered, in which the shape and bandwidths of individual kernel functions are locally adapted to their position (e.g., Vieu, 1991).

It is interesting to note that the estimate of the conditional mean provided by the kernel method has a form of a local average of the training samples (y^i)

$$\hat{m}(\mathbf{x}) = \sum_{i=1}^{T} w_i(\mathbf{x}) \cdot y^i \text{ (where } w_i(\mathbf{x}) \text{ decreases with } \|\mathbf{x} - \mathbf{x}^i\|) \tag{13}$$

which minimizes the weighted least-squares error to the training set (Härdle, 1990), i.e.,

$$\hat{m}(\mathbf{x}) = \min_{\theta} \sum_{i=1}^{T} w_i(\mathbf{x})(\theta - y^i)^2. \tag{14}$$

A direct method of implementing the GRNN would be to store the whole training set, choose a kernel function and its smoothness based on the characteristic of the training set, and finally use the network by evaluating the kernel functions at each training sample for a particular value of the input, $\mathbf{x}$, and producing a weighted sum of the training responses, y^i. Unfortunately, for a large training-set size this may prove computationally expensive, and the network size has to be reduced, usually by using various clustering techniques (Specht, 1991).

In Section 4 we show how a specifically modified NTNN implicitly realizes a regression estimator.

3. THE APPROXIMATION-TYPE NTNN

Some details concerning the structure and operation of the NTNN vary depending on the particular network application. Here, we are mainly concerned with a variant of the architecture suitable for approximating arbitrary smooth and bounded $\mathscr{R}^D \rightarrow \mathscr{R}$ functions, defined on a compact domain. The network consists of an R-bit binary array (traditionally called a retina) and a set of K memory nodes, each having a N-bit long address word (i.e., having 2^N addressable locations), where each memory location is assumed to have a real-number format. The $\mathscr{R}^D \rightarrow \mathscr{R}$ mapping performed by the network consist essentially of three stages:

1. Conversion of the real vector input into a binary format and projecting it onto the network retina.
2. Sampling of the retina by a set of K-tuple memory nodes, each forming its address with N randomly selected array bits, from the R total retina bits available.
3. Combining the contents of the addressed memory locations (by summation) to produce the network response.

The vector-to-binary conversion should provide a unique retina pattern for every possible network input. However, because the size of the array is finite, a limitation on the input variable resolution has to be imposed. Normally, the vector coordinates are quantized (e.g., linearly) to an integer format which lends itself more directly to a binary mapping. It is further assumed that the coordinates are normalized, so that each results in an integer variable within a uniform range. A common conversion procedure can be applied to map the values of these coordinates into binary patterns, each pattern having the same size. Thus each input vector variable is mapped onto R/D bits of the retina array.

Once the retina pattern has been obtained, the sampling process assigns to it a unique set of K binary N-tuples

$$\{t_1(\mathbf{x}), t_2(\mathbf{x}), \ldots, t_K(\mathbf{x})\}. \tag{15}$$

As each N-tuple is naturally associated with an integer in the range of $0, \ldots, 2^N - 1$, the symbol $t_k(\mathbf{x})$ corresponds to the value (or index) of the tuple address selected for the kth memory node by a pattern corresponding to input $\mathbf{x}$. Normally, the sampling algorithm and the number of tuple memory nodes are chosen such that every retina bit is a member of at least one N-tuple (unless memory limitations impose an undersampling condition).

As each memory location contains a real number, any particular choice of tuple addresses results in a selection of K numerical weights

$$\{w_1(\mathbf{x}), w_2(x), \ldots, w_K(\mathbf{x})\}. \tag{16}$$

These are summed to give the network response for the particular input, $\mathbf{x}$. Thus the approximation-type NTNN follows a single-layer paradigm (the hidden layer being implicitly realized by N-tuple sampling); and the LMS algorithm (Widrow & Stearns, 1985) is usually used for weight update during the training (similarly to other single-layer architectures, e.g.,

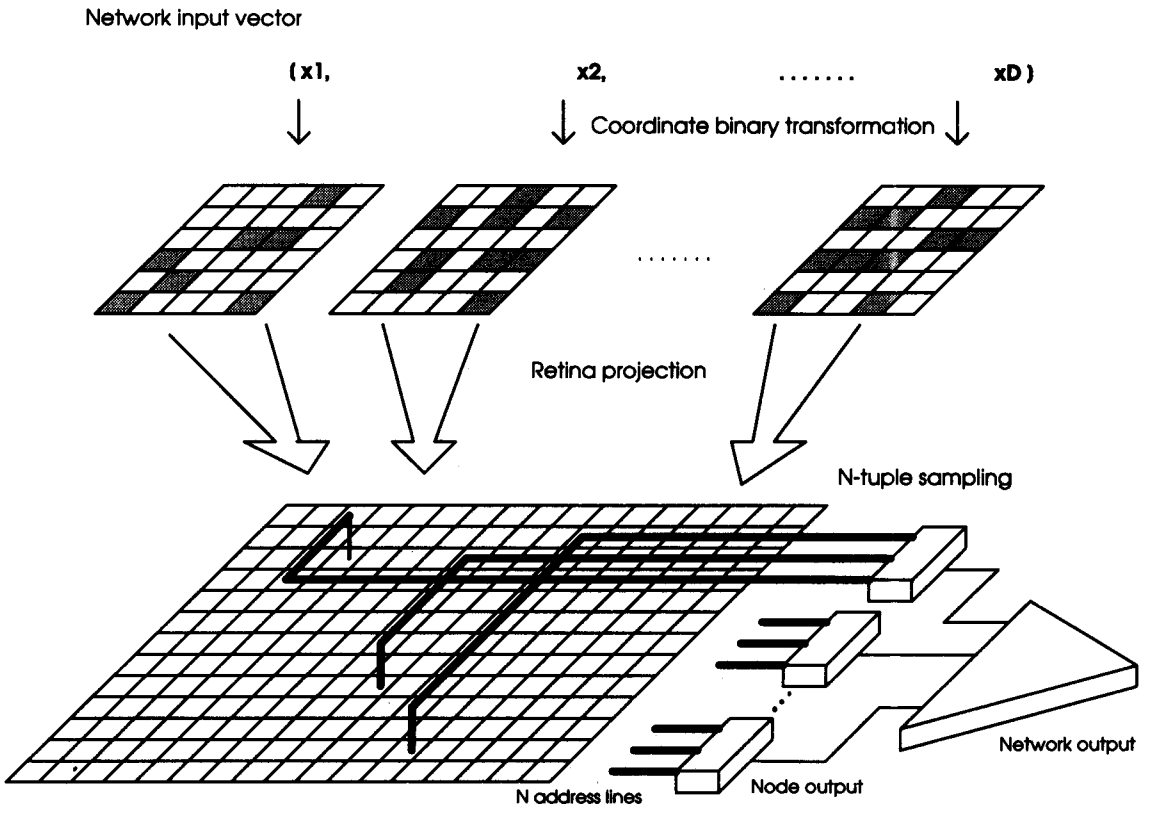

FIGURE 1. Stages involved in the approximation-type NTNN mapping.

Adaline, CMAC, RBF). Figure 1 shows the steps involved in the NTNN mapping.

To analyse the generalization properties of this network it is useful to define the tuple distance between two network inputs as the number of different tuple addresses they generate:

$$\rho(\mathbf{x},\mathbf{z}) = \sum_{k=1}^{K} (t_k(\mathbf{x}) \neq t_k(\mathbf{z})) \quad \rho(\cdot\cdot) \in \{0,1,\ldots,K\}. \quad (17)$$

It is clear, that as long as every possible input to the network results in a unique retina pattern, the tuple distance is equivalent to a generalized Hamming distance, and thus satisfies all the metric function conditions (i.e., identity, symmetry, and the triangle inequality). For correct operation the value of this function should be proportional to the input-space distance or, more specifically, there should be a monotonically increasing relationship between the input and tuple distances. In this way the topological properties of the input space can be reflected in the space of binary patterns.

The problem of finding the dependence between the input and tuple distance functions can be approached more conveniently if two separate relationships are combined

—in the input distance → pattern distance (i.e., Hamming metric) mapping;

—the pattern distance → tuple distance mapping.

Several analyses of the tuple distance have been carried out (Johnson, 1991; Allinson & Johnson, 1993; Tattersall et al., 1991). It has been shown that the expected value of the tuple distance corresponding to the Hamming distance of H is given by (Kolcz & Allinson, 1995)

$$E(\rho(\mathbf{X},\mathbf{Z})) = K\left(1 - \left(1 - \frac{H}{R}\right)^N\right)$$
$$\approx K(1 - \exp(-N \cdot h)) \text{ where } h - H/R. \quad (18)$$

The approximate exponential relationship, $E(\rho(\mathbf{X},\mathbf{Z})) \approx K(1 - \exp(-N \cdot h))$, has also been suggested in Tattersall et al. (1991) and Johnson (1991). From the above relationship it is apparent that the network is inherently sensitive to the Hamming distances between retina patterns, and it is therefore advantageous to achieve proportionality between the input and pattern distances. Since the input vector coordinates are usually transformed individually into a binary format and then merged together to form a complete retina pattern, the natural proportionality can be sought between the L_1 (city-block) input metric and the Hamming metric. Full proportionality is possible when a thermometer code (Aleksander et al., 1984) is used for coordinate mapping (so called, because it encodes a positive

integer of value q by setting q consecutive bits of a binary array to 1, which is analogous to the mercury level in a thermometer), in which case the size in bits of a binary code-word associated with an integer variable of n bits is 2^n in length. Unfortunately, if the input quantization provides large ranges of input coordinates (together with a large number of input dimensions), system limitations may render the thermometer code impractical. Several other coding schemes have been proposed, all of which result in smaller retina sizes at the cost of less regular relationships between the L_1 and Hamming distances (Tattersall et al., 1991; Kolcz & Allinson, 1994). Specifically, the use of the CMAC (Albus, 1975; Kolcz & Allinson, 1994, 1995) code offers the proportionality between the L_1 and Hamming distances up to a fraction of the available range, which can be easily controlled. In the following discussions the use of the thermometer code is assumed (i.e., $L_1(\mathbf{x},\mathbf{z}) = H(\mathbf{x},\mathbf{z})$).

4. REGRESSION TRAINING ALGORITHM FOR THE NTNN

To realize the regression network, the general structure of the approximation network presented above is modified such that each tuple memory location stores an integer counter value apart from the real-valued weight. Let $a_k(\mathbf{x})$ designate the counter value corresponding to the location addressed in the kth tuple memory by the input $\mathbf{x}$. Thus any input to the network results in a unique selection of K-tuple addresses together with their associated weight and counter values

$$\mathbf{x} \rightarrow \begin{cases} \{t_1(\mathbf{x}), t_2(\mathbf{x}), \ldots, t_K(\mathbf{x})\} \\ \{w_1(\mathbf{x}), w_2(\mathbf{x}), \ldots, w_K(\mathbf{x})\} \\ \{a_1(\mathbf{x}), a_2(\mathbf{x}), \ldots, a_K(\mathbf{x})\}. \end{cases} \tag{19}$$

Initially, all network tuple memory locations (both the weight and counter values) are set to zero. During the training phase the network is presented with T training pairs $(\mathbf{x}^i, y^i)$ drawn according to the pdf of the system being modelled, where $\mathbf{x}^i$ is the D-dimensional input vector, and y^i denotes the corresponding output. For each tuple location addressed by $\mathbf{x}^i$ the value of y^i is added to the corresponding weight, and the location counter is incremented:

$$\begin{aligned} &w_k(\mathbf{x}^i) \leftarrow w_k(\mathbf{x}^i) + y^i \quad \text{and} \\ &a_k(\mathbf{x}^i) \leftarrow a_k(\mathbf{x}^i) + 1 \quad i = 1, \ldots, T \quad k = 1, \ldots, K. \end{aligned} \tag{20}$$

During the recall phase the network output, $\hat{y}(\mathbf{x})$, is obtained by normalizing the sum of addressed weights with the sum of their corresponding counter values. In cases where all addressed counter locations are zero (i.e., none of the selected tuples has been encountered during the training), the output is set to zero

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{x})}{\sum_{k=1}^{K} a_k(\mathbf{x})} \qquad \sum_{k=1}^{K} a_k(\mathbf{x}) = 0 \rightarrow \hat{y}(\mathbf{x}) = 0. \tag{21}$$

It remains to be shown that the function implemented by the network achieves an approximation of the regression function, $E(Y/\mathbf{x})$, using a valid kernel for the pdf estimate.

4.1. Derivation of the Regression Equation

Let $\mathbf{x}$ denote an arbitrary input presented for a recall after the network has been trained on T pairs $(\mathbf{x}^i, y^i)$. After the retina mapping $\mathbf{x}$ results in the selection of a particular set of tuples

$$\{t_1(\mathbf{x}), t_2(\mathbf{x}), \ldots, t_K(\mathbf{x})\} \tag{22}$$

and their associated contents. The value of each addressed weight is given by the sum of all values, y^i, for which the training input vector, $\mathbf{x}^i$, resulted in the selection of the tuple location containing the weight. Thus

$$\begin{aligned} &w_k(\mathbf{x}) = \sum_{i=1}^{T} y^i \cdot M_k^i(\mathbf{x}) \\ &\text{where} \quad M_k^i(\mathbf{x}) = \begin{cases} 1 & \text{if } t_k(\mathbf{x}) = t_k(\mathbf{x}^i) \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{23}$$

Note that since the tuple distance between two inputs to the network is defined as the number of different tuple addresses they generate, the following relationships are true

$$\begin{aligned} &\sum_{k=1}^{K} M_k^i(\mathbf{x}) = K - \rho(\mathbf{x}, \mathbf{x}^i) = K \cdot \left(1 - \frac{\rho(\mathbf{x}, \mathbf{x}^i)}{K}\right) \\ &\text{where} \quad \rho(\cdot,\cdot) = 0, 1, \ldots, K \end{aligned} \tag{24}$$

$$\sum_{i=1}^{T} M_k^i(\mathbf{x}) = a_k(\mathbf{x}) \quad \text{where} \quad a_k(\mathbf{x}) = 0, 1, \ldots, T. \tag{25}$$

Therefore, the sum of weights selected by the input vector $\mathbf{x}$ can be expressed as

$$\sum_{k=1}^{K} w_k(\mathbf{x}) = \sum_{k=1}^{K}\sum_{i=1}^{T} y^i \cdot M_k^i(\mathbf{x}) = \sum_{i=1}^{T} y^i \sum_{k=1}^{K} M_k^i(\mathbf{x})$$
$$= K\sum_{i=1}^{T} y^i \cdot \left(1 - \frac{\rho(\mathbf{x}, \mathbf{x}^i)}{K}\right) \tag{26}$$

and the sum of the corresponding counter values is given by

$$\sum_{k=1}^{K} a_k(\mathbf{x}) = \sum_{k=1}^{K}\sum_{i=1}^{T} M_k^i(\mathbf{x}) = \sum_{i=1}^{T}\sum_{i=1}^{K} M_k^i(\mathbf{x})$$
$$= K\sum_{i=1}^{T}\left(1 - \frac{\rho(\mathbf{x}, \mathbf{x}^i)}{K}\right). \tag{27}$$

The output of the network, given by the ratio of the above two sums, is thus

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{x})}{\sum_{k=1}^{K} a_k(\mathbf{x})} = \frac{\sum_{i=1}^{T} y^i \cdot \left(1 - \frac{\rho(\mathbf{x}, \mathbf{x}^i)}{K}\right)}{\sum_{i=1}^{T}\left(1 - \frac{\rho(\mathbf{x}, \mathbf{x}^i)}{K}\right)} \tag{28}$$

which provides an approximate solution to the regression function $E(Y/\mathbf{x})$ provided that

$$\Phi(\cdot,\cdot) = \left(1 - \frac{\rho(\cdot,\cdot)}{K}\right)$$

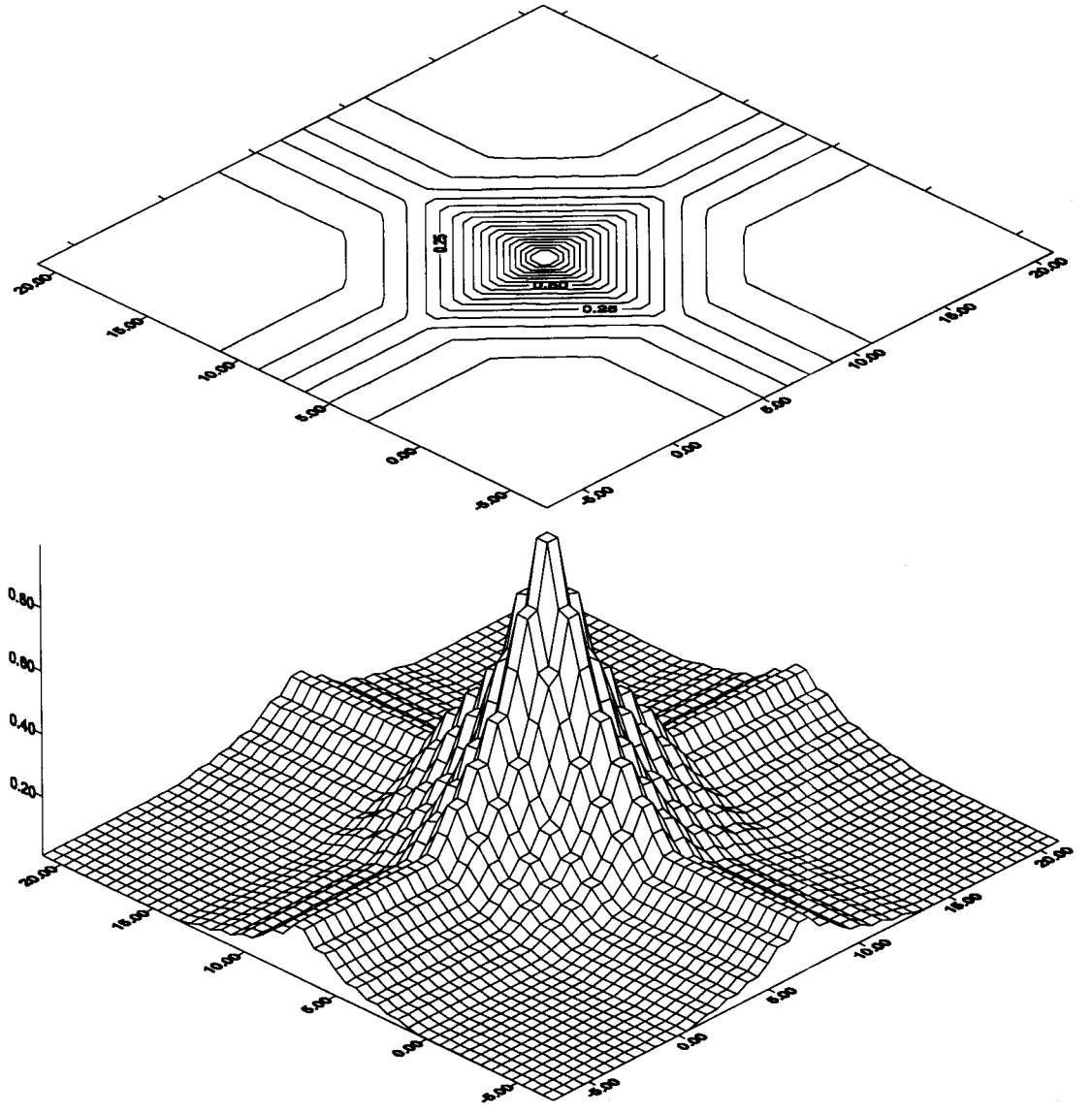

FIGURE 2. The tuple kernel function for a two-dimensional case with $N = 6$. The input variables are uniformly quantized within the [0, 14] interval and are clipped to 0 and 14 below and above this range, respectively. The kernel is centred about the point (7, 7) and a region corresponding to [−7,21]×[−7,21] is shown; saturation regions of the kernel function are visible.

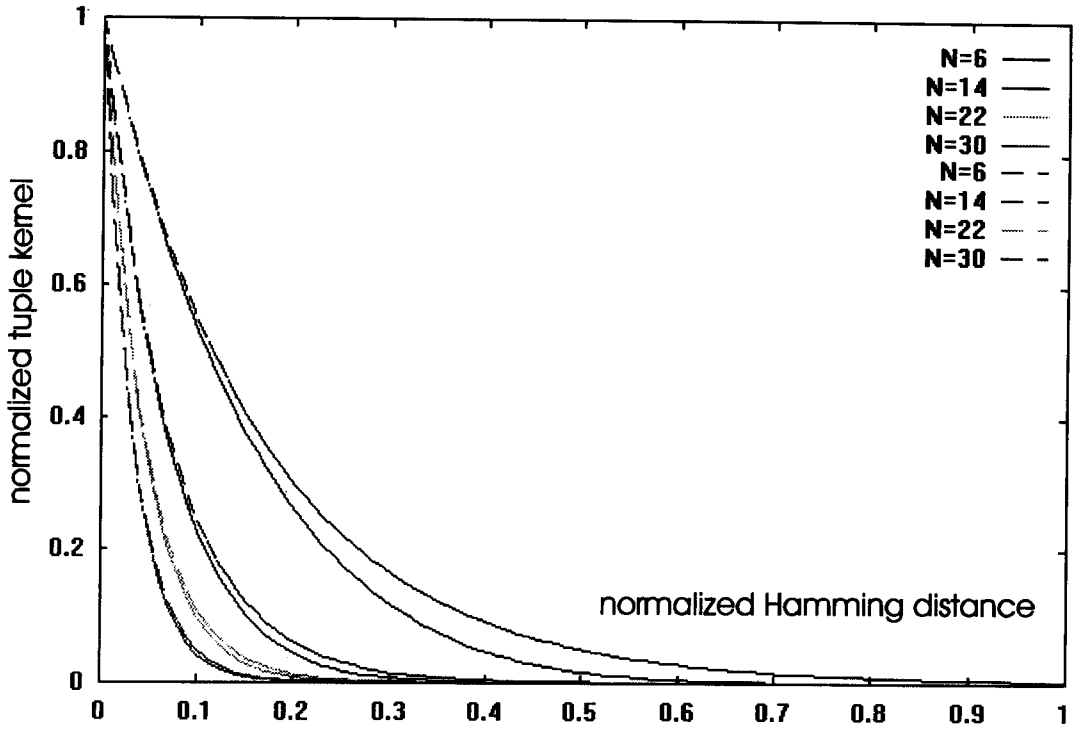

FIGURE 3. Tuple kernel function and its exponential approximate (less steep) for a range of smoothing parameters, *N*.

is a valid kernel function. According to eqn (18) and taking the input quantization into account (assuming the thermometer code for vector coordinate mapping) the kernel function is given by

$$\Phi(\mathbf{x},\mathbf{z}) = \begin{cases} 1 - \frac{\rho(\mathbf{x},\mathbf{z})}{K} = 1 - \left(1 - \left(1 - \frac{|\mathbf{x}-\mathbf{z}|}{R}\right)^N\right) \\ = \left(1 - \frac{|\mathbf{x}-\mathbf{z}|}{R}\right)^N \quad \text{for} \quad 0 \leqslant |\mathbf{x}-\mathbf{z}| < R \\ 0 \quad \text{for} \quad |\mathbf{x}-\mathbf{z}| \geqslant R \end{cases} \tag{29}$$

where $|\mathbf{x}-\mathbf{z}| = L_1(\mathbf{x},\mathbf{z})$. It has to be noted that since each vector coordinate, x_d, is quantized to a finite number of levels, a clipping of the norm function, $|\cdot|$, occurs. Let $|\cdot|_R$ denote the value of the L_1 norm before the quantization. Assuming that the vector coordinates are quantized to the same number of levels, Q, the clipped norm can be expressed as

$$|\mathbf{x}-\mathbf{z}| = \sum_{d=1}^{D} |x_d - z_d|$$

$$\text{where} \quad |x_d - z_d| = \begin{cases} |x_d - z_d|_R & \text{if} \quad |x_d - z_d|_R < Q \\ Q & \text{if} \quad |x_d - z_d|_R \geqslant Q \end{cases} \tag{30}$$

Thus effectively, $|\cdot| = |\cdot|_R$ is satisfied inside the hypercube $[0,Q]^D$ (provided that the coordinates are quantized to positive integers) and saturates to the clipped values outside this region. Figure 2 shows these saturation effects for a two-dimensional case.

The function $\Phi(\cdot,\cdot)$ is continuous, symmetrical, non-negative and has finite support. Its value depends (i.e., it is monotonically decreasing) only on the distance between its arguments (i.e., it is position independent). After normalization to give a unity integral over its domain, $\Phi(\cdot,\cdot)$ is a valid density function and can be used as an estimation kernel. Although this function is not directly representable as a product of univariate kernel functions, its close approximation is, since

$$\Phi(\mathbf{x},\mathbf{z}) = \begin{cases} \left(1 - \frac{|\mathbf{x}-\mathbf{z}|}{R}\right)^N \approx \exp\left(-N \cdot \frac{|\mathbf{x}-\mathbf{z}|}{R}\right) \\ = \prod_{d=1}^{D} \exp\left(-N \cdot \frac{|x_d - z_d|}{R}\right) \\ \text{for} \quad 0 \leqslant |\mathbf{x}-\mathbf{z}| < R \\ 0 \text{ for } |\mathbf{x}-\mathbf{z}| \geqslant R \end{cases} \tag{31}$$

where the kernel smoothing parameter is given by R/N. The exponential function satisfies all of the univariate kernel function conditions (Parzen, 1962; Hand, 1982). Figure 3 compares the one-dimensional N-tuple kernel function and its exponential approximate for a range of N (the distances are normalized to lie inside the unity interval).

4.2. Advantages of the *N*-tuple Regression Network

One major advantage of the N-tuple implementa-

tion is the speed of operation. Since each network recall involves addressing K locations (a fixed number), the response does not depend on the number of $(\mathbf{x}^i, y^i)$ pairs with which the network has been trained. As a result, there is no need for explicit data clustering, often used in conventional regression architectures or the radial basis function network (e.g., Moody & Darken, 1989) for large training sizes. The mapping performed by the NTNN consists mainly of combinatorial sampling and simple memory look-up operations, and thus can be realized very efficiently, especially in hardware (e.g., Aleksander et al., 1984).

5. NTNN IMPLEMENTATION ISSUES

5.1. An Optimum Choice of the Kernel Smoothing Parameter N

As for other regress-type networks, the bandwidth of the kernel function has to be chosen according to the training set characteristics. In the case of the N-tuple network the kernel function depends on the sample size, N, and becomes narrower with increasing N. The existing methods for selecting the kernel function bandwidth can be used for the "optimum" choice of N (Specht, 1991; Scott, 1992). For example, one might use a cross-validation technique, where the network is trained using only a subset of the complete training set and an error function (usually a squared error) produced by the network for the remainder of the training points is used as a validation criterion. One of the more popular variants of this method relies on creating the network using all the training points available, except one, which is then used for error estimation (i.e., the leave-one-out method). This process is repeated for all the training points to give a total error for a given choice of the smoothing parameter. An optimization algorithm can be then used to determine the minimum of the error curve.

Since N can only take discrete values, the number of choices is smaller than for continuous smoothing parameters and this can accelerate the selection process. As other experiments with the NTNN suggest, the dependence of the network on tuple size is not very sharp (e.g., Aleksander & Stonham, 1979), and there will usually be a range of values of N which provide a comparable performance. A similar relative insensitivity of the GRNN to variation of the smoothing parameter around the optimum has been observed by Specht (1991).

5.2. Dealing with Large Values of N

For cases where the distribution of the training samples is very localized in the input space, and also when the size of the training set is large, the bandwidth selection process can lead to very large values of N (i.e., very narrow kernels). This may result in excessive memory requirements, as well as in high probability of accessing empty locations by input points from outside the training set.

It is obvious that for large values of N each memory-node would have to consist of a very large number of locations. However, during the network training only a small fraction of possible memory addresses is actually encountered. Thus hash-memory techniques (Knuth, 1973) are appropriate for realizing efficient weights/counters storage. Several of the hashing techniques have also been proposed for custom hardware implementations (cf., a review in Kohonen, 1984).

Large values of N usually correspond to a high density of input points being used as a training set. Since the input points which are used during the normal network operation (e.g., testing) come from the same distribution, they should also be located close to many points from the training set. Hence there will be a high degree of overlap between the corresponding retina patterns. However, since the N-tuple sampling is a statistical technique where each sample corresponds to a small feature of the retina pattern, the size of the retina should be increased accordingly so that the relationship $R \gg N$ is valid. On the other hand, the actual kernel function utilized by the network agrees with formula (29) only in an average sense and will usually be rather irregular if the number of tuples, K, is small (Kolcz & Allinson, 1995). Therefore, K should be made at least large enough to allow complete sampling of the retina, and generous oversampling is desirable for improving the approximation capabilities. One possible method of optimizing the values of N and R is to select the value of R based on the range and variance parameter of the data set and then select K so that the error level provided by the NTNN is as close as possible to that generated for a GRNN with a simulated tuple kernel function. Of course, both the increase in the retina size and the number of memory nodes have performance and resources implications, especially in sequential computer simulations, and in practice some trade-off is always necessary.

5.3. Data Normalization

Since the kernel function is symmetrical, the same degree of smoothing occurs along each dimension. Consequently, it is desirable that the input coordinates are normalized to the same range and variance (Specht, 1991). The choice of the number of quantization steps is affected by the range and variance of normalized input vector coordinates. In particular, a smaller number of quantization levels for low-resolution variables could be chosen, with the

corresponding binary patterns being expanded to the length used by other variables. One possible method of normalizing a D-dimensional input vector consists of the following steps (i.e., after establishing the minimum/maximum bounds and variance for each vector dimension):

1. Rescaling each real vector variable to the [0, 1] range.
2. Converting it to an integer value using a linear quantizer with the maximum of Q quantization steps, where Q satisfies the condition

$$Q > \max_d \frac{range(x_d)}{\sigma_d} \quad d = 1, \ldots, D$$

in which case the change by a standard deviation in any real input variable will change the value of the corresponding integer variable at least by one.

The value of Q may have to be increased for large values of N as explained in Section 5.2, as the retina size (assuming thermometer code) is given by $R = D \cdot Q$, and it is desirable that $R \gg N$.

6. EXPERIMENTAL RESULTS

To demonstrate the interpolation capabilities of the N-tuple network as well as to compare it with other architectures two test tasks have been investigated. In both cases the thermometer code has been applied to transform the input variables into binary retina patterns.

6.1. Example 1: Non-linear Plant Approximation

After Specht (1991) the non-linear function

$$g(x_1, \ldots, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^2 + x_3^2} \tag{32}$$

provides a simulation of a plant governed by the following recurrence equation

$$y_p(k+1) = g\big(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)\big) \tag{33}$$

where $y_p(k)$ and $u(k)$ represent the output and the input of the plant at the kth time step, respectively. The network was trained on 1000 sample pairs, where the control input $u(k)$ had been drawn from a uniform distribution in the $[-1, 1]$ interval while the remaining variables were calculated using the recurrence equation (33), with zero initial conditions. The test set consisted of 1000 points generated by choosing the control input according to

$$u(k) = \begin{cases} \sin(2\pi k/250) & \text{for } k \leqslant 500 \\ 0.8 \cdot \sin(2\pi k/250) + 0.2 \cdot \sin(2\pi k/25) & \text{for } k > 500. \end{cases} \tag{34}$$

Apart from the regression-type NTNN two other networks have been considered:

—a GRNN with a simulated N-tuple kernel given by eqn (29)

—a GRNN with the Gaussian kernel,

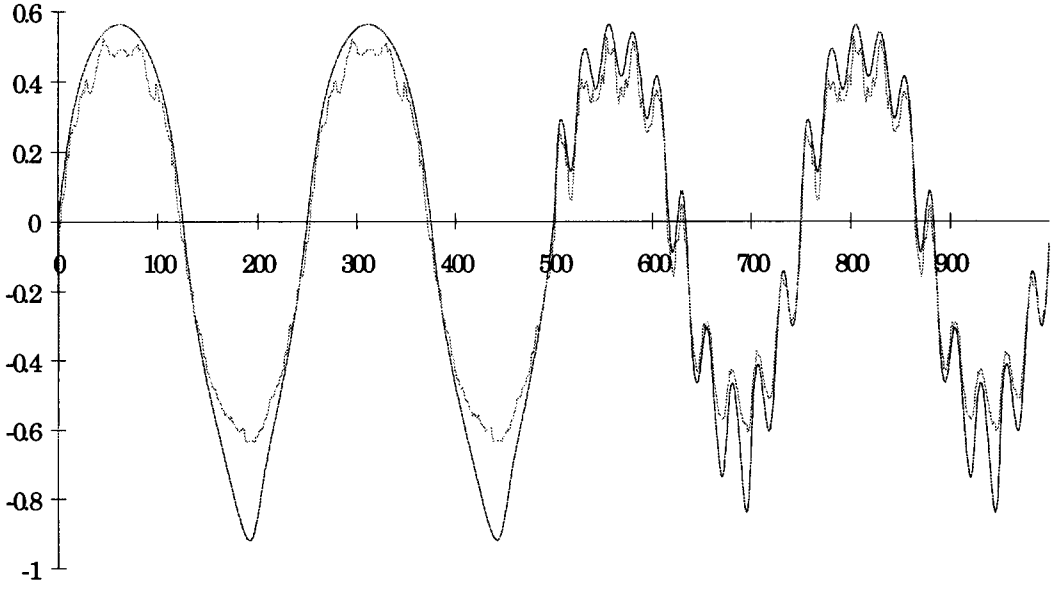

FIGURE 4. Simulation run of the regression NTNN for $N = 31$, $Q = 512$, and $T = 1000$ training points.

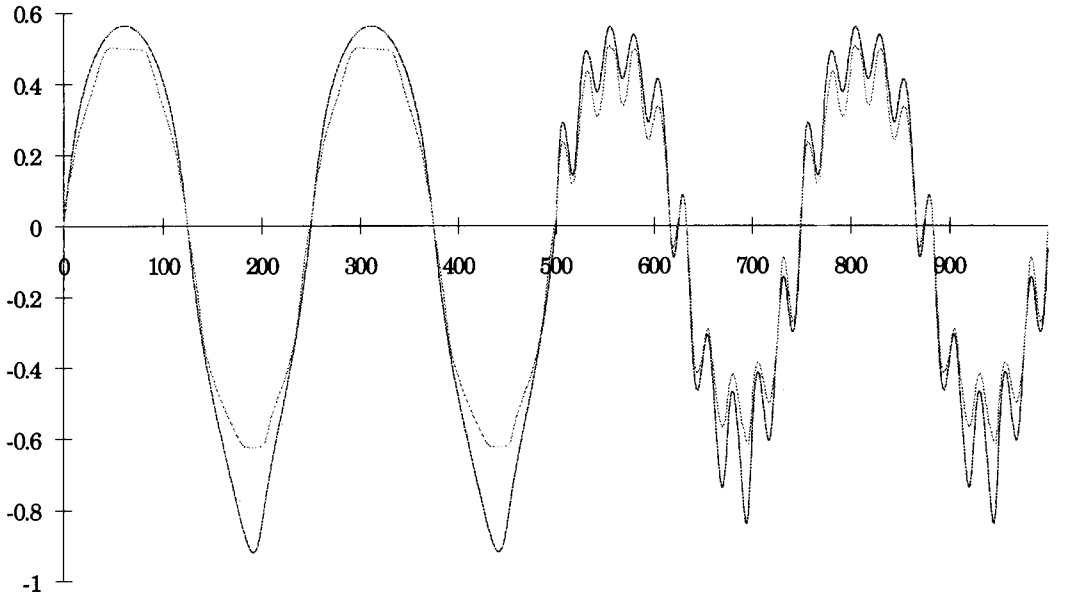

FIGURE 5. Simulation run of the GRNN with an *N*-tuple kernel function, eqn (29), for $N = 31$ and $T = 1000$ training points.

$$G(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right)$$

($\|\cdot\|$ denotes Euclidean norm in the input space). (35)

The NTNN used a tuple size of $N = 31$, the input vector variables were quantized to $Q = 512$ levels; whereas a smoothing parameter of $\sigma = 0.315$ was used for the Gaussian kernel, which provides approximately the same neighbourhood size as the value of N chosen for the NTNN kernel function.

Figures 4 and 5 show the simulation results for the NTNN and GRNN with a simulated N-tuple kernel. It is clear that both networks provide a very similar approximation performance. The slightly more irregular waveforms generated by the NTNN can be attributed to the dependence of the actual tuple distance function on both of its arguments, rather than only on their distance.

Figure 6 gives the approximation provided by the GRNN with a Gaussian kernel; and although the Euclidean symmetry of the Gaussian slightly improves the approximation quality, all of the networks considered result in comparable performance for this task.

As previously noted, the distance function realized

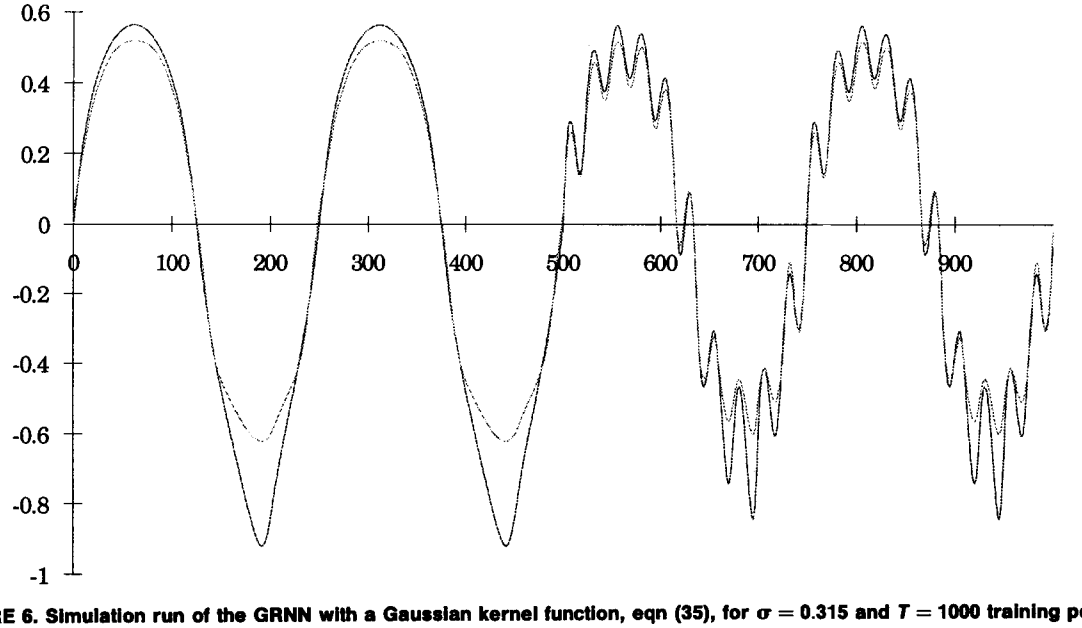

FIGURE 6. Simulation run of the GRNN with a Gaussian kernel function, eqn (35), for $\sigma = 0.315$ and $T = 1000$ training points.

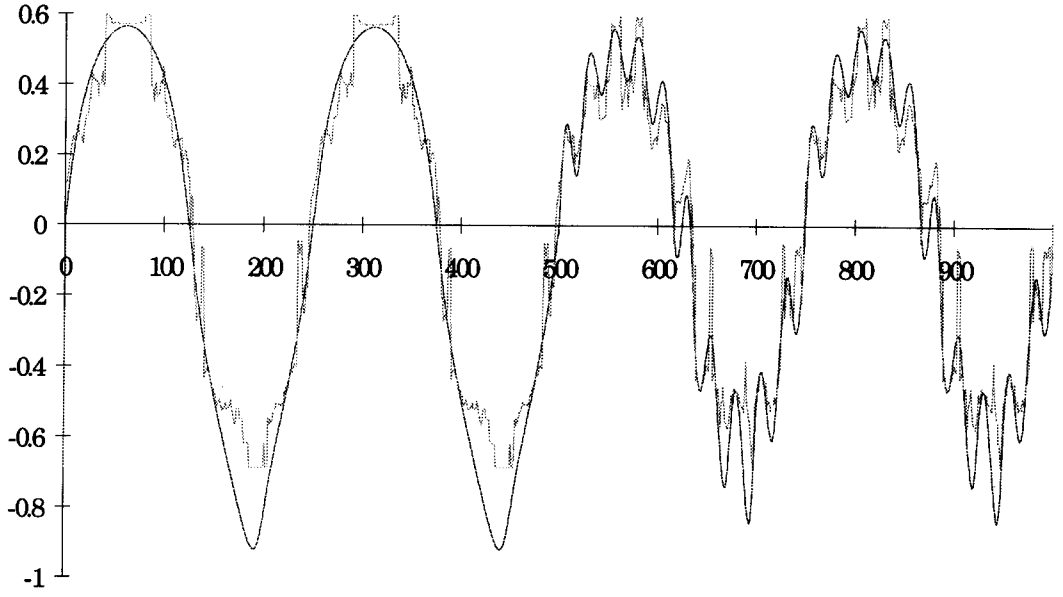

FIGURE 7. Simulation run of the regression NTNN for $N = 31$, $Q = 64$, four-fold oversampling, and $T = 1000$ training points.

by the network is accurate only in the statistical average sense and its value for a particular pair of inputs becomes more erratic when the number of tuples is small. This becomes especially significant for large values of N and a small number of retina bits, R. To demonstrate this we present the network simulation results on the same training set but with the number of quantization steps per coordinate reduced from 512 to 64. In one case the number of tuples allows each retina bit to be sampled just once, whereas the second case uses four-fold retina oversampling. From Figures 7 and 8, note that the increase in the number of tuples taken results in a much smoother estimate.

TABLE 1
Choice of the Bandwidth for the GRNN with Gaussian and *N*-tuple Kernels

Training Set Size	Optimum σ	Optimum N	Sub-optimum N
100	0.038	168	20
1000	0.020	257	140
5000	0.011	485	280
10,000	0.011	537	

6.2. Example 2: Mackey–Glass Chaotic Time Series Prediction

The chaotic time series, given by a differential delay equation

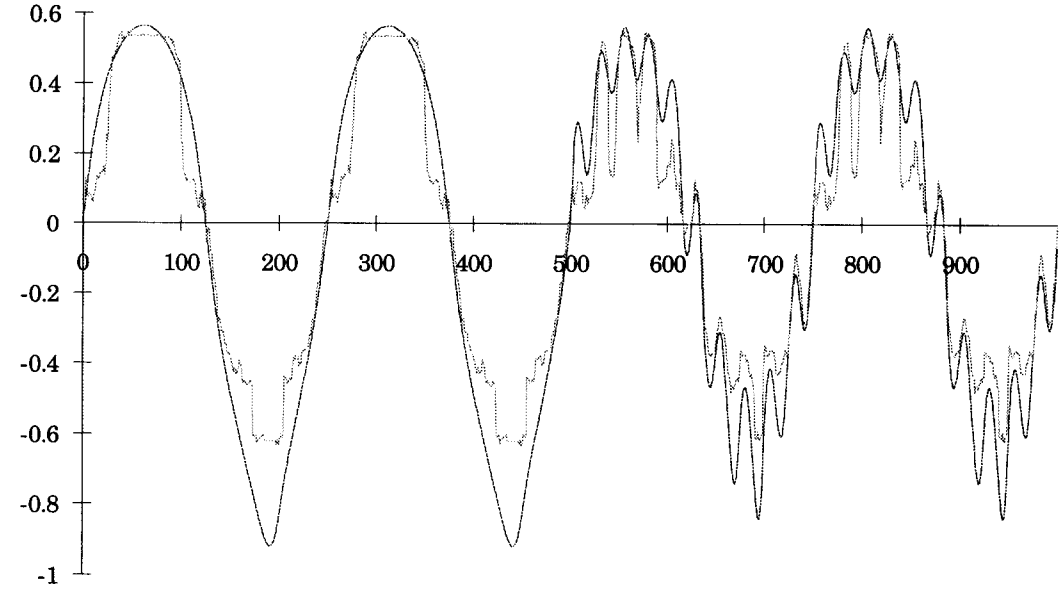

FIGURE 8. Simulation run of the regression NTNN for $N = 31$, $Q = 64$, four-fold oversampling, and $T = 1000$ training points.

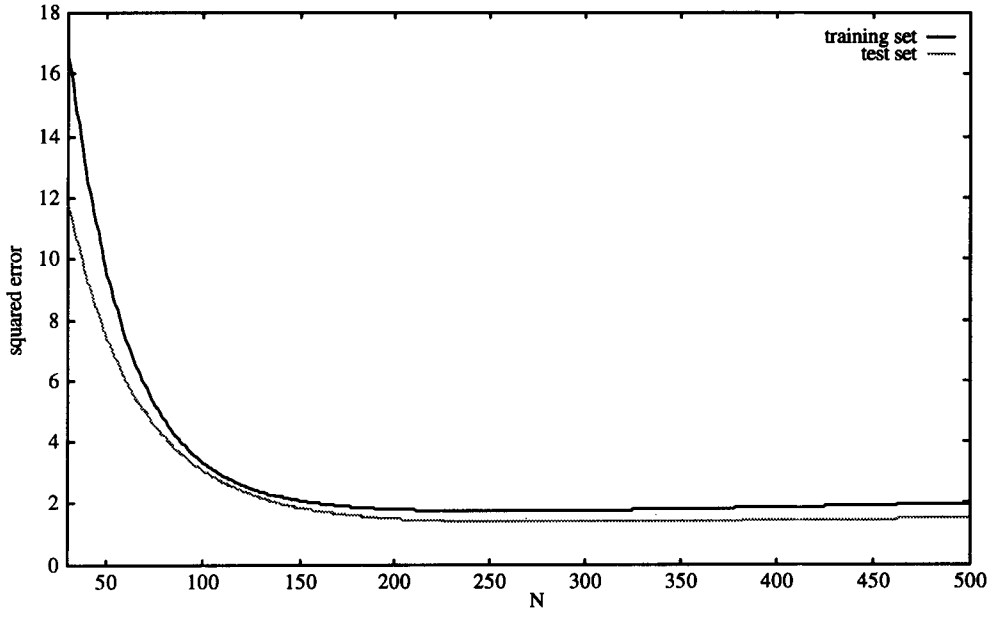

FIGURE 9. Mackey–Glass time series prediction by the regression NTNN: Dependence of the approximation squared error on the value of *N*, for both the training and test sets (trained on 1000 and tested on 500 points).

$$\frac{dx(t)}{dt} = -b \cdot x(t) + a \cdot \frac{x(t-\tau)}{1 + x(t-\tau)^{10}} \tag{36}$$

has been studied extensively (Lapedes & Farber, 1987) and used as an example for demonstrating neural networks approximation capabilities (e.g., Moody & Darken, 1989; Platt, 1991). After Moody and Darken (1989) we set the parameters to $\tau = 17$, $a = 0.2$, $b = 0.1$, which results in a characteristic time of the series of $t_{char} \approx 50$. The problem of estimating the function

$$x(t+F) = g(x(t), x(t-\Delta), x(t-2\Delta), x(t-3\Delta)),$$
$$\text{with} \quad \Delta = 6 \text{ and } F = 85 > t_{\text{char}} \tag{37}$$

was considered. The number of training samples was varied between 100 and 10000, and the normalized

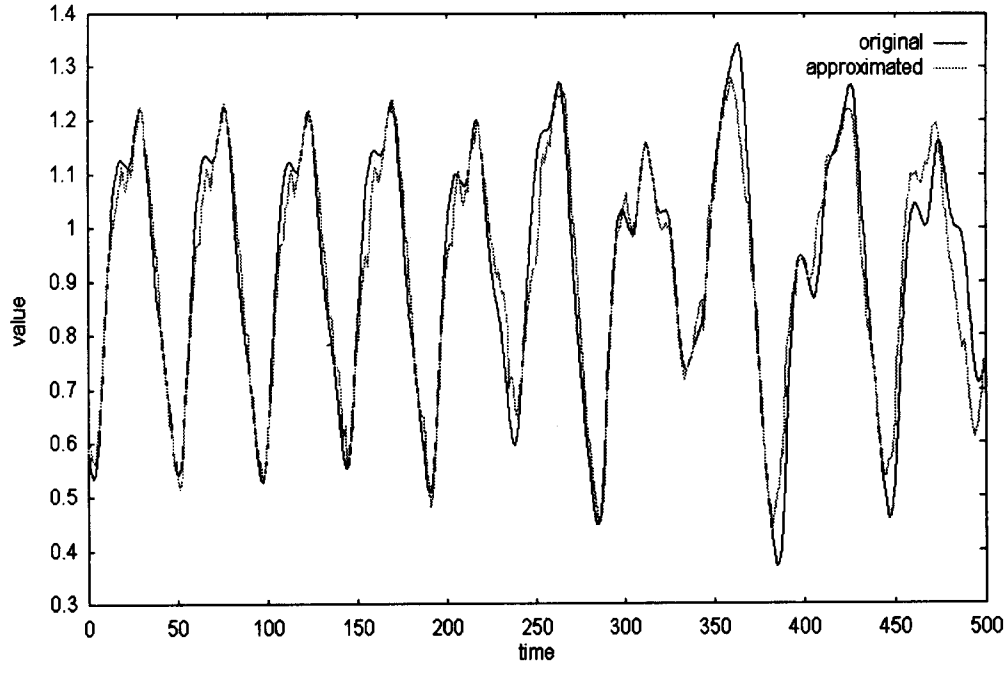

FIGURE 10. Simulation run of the regression NTNN for $N = 140$, $Q = 2000$, $K = 140$, and $T = 1000$ for the Mackey–Glass chaotic time series prediction.

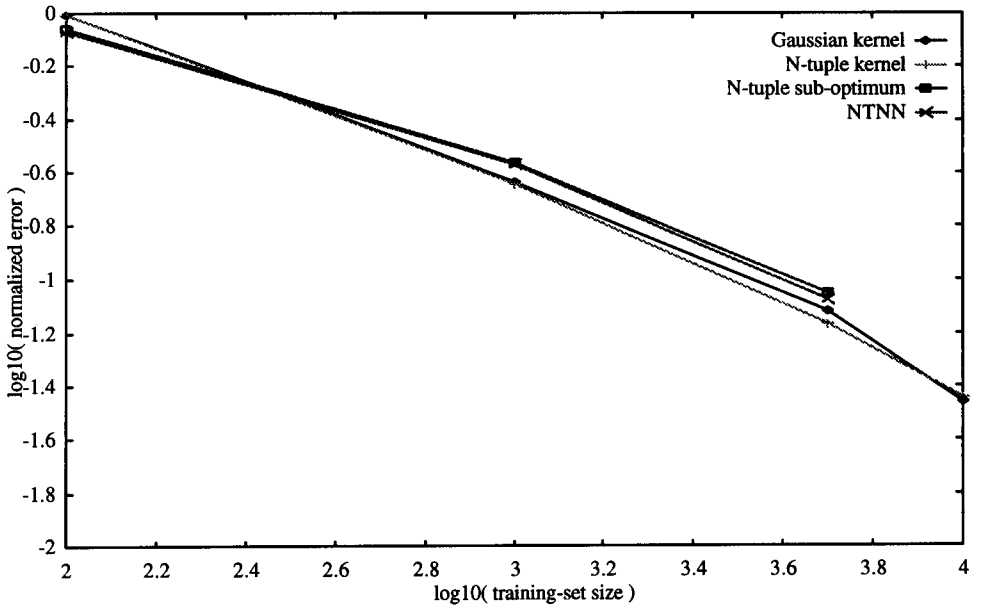

FIGURE 11. Normalized approximation errors for the Mackey–Glass time-series prediction generated by the GRNN with Gaussian and *N*-tuple kernels as well as by the regression NTNN.

prediction error, defined as

$$\sqrt{\frac{\sum_{test\ set}(x(t)-\hat{x}(t))^2}{\sum_{test\ set}(x(t)-E(x(t)))^2}} \tag{38}$$

where $\hat{x}(t)$ denotes the network estimate of $x(t)$, was used as a measure of the network performance. In all cases the test set consisted of 500 points. As with the previous example, we considered the GRNN with Gaussian and N-tuple kernel functions as well as a full implementation of the NTNN. The "leave-one-out" method of the kernel bandwidth selection gave the results shown in Table 1 (columns 2 and 3). It is apparent that optimization produced very large values of N. However, as the error curves tend to be very flat around the minimum, much smaller values of N can be chosen in practice. As an example, Figure 9 shows the squared error for both the training and test sets as a function of N for the case of 1000 training samples. It can be seen that the function drops sharply in the region of small N and then stabilizes as N increases. Additionally, it is seen that the optimum bandwidth given by the "leave-one-out" method in this case produces a value very close to that for minimizing the error over the entire test set. Figure 10 shows the actual and predicted values of the series for the $T = 1000$ case, and for a NTNN with $N = 140$. Figure 11 gives the normalized error as a function of the training set size for all the networks considered. As can be seen, there is little difference between the results obtained by the GRNN with Gaussian and N-tuple kernels, and the NTNN provides a performance close to the N-tuple GRNN prediction (the smaller, sub-optimum values of N given in Table 1 did not significantly affect the network performance). The apparent linear relationship between the normalized error and the training-set size agrees with the results reported by Moody & Darken (1989) for an RBF network.

7. CONCLUSIONS

A modification of the standard NTNN, allowing an implementation of the GRNN, has been proposed. The network employs a valid kernel function whose "optimum" smoothing parameters can be found using existing heuristics. This particular realization of the GRNN permits efficient system design with predictable and constant response times, independent of the amount of training involved. It allows the direct use of large training sets and does not require an application of data clustering techniques.

Additionally, since the distances between inputs to the network and the training points are computed implicitly, the network provides very fast response times, determined essentially by the time necessary to convert the input into a binary form, perform a fixed number of memory look-up operations and carry out the final summation.

From the point of applying the N-tuple architecture to function approximation problems, the regression training compares favourably with the LMS training, which usually requires more passes through the training set before the learning achieves sufficiently low error levels. However, this modification of the NTNN requires more memory storage, as each location contains a counter value as well as a weight value.

REFERENCES

Albus, J. S. (1975). A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC). *J. Dynam. Syst. Measurement Control*, **97**(3), 220–227.

Aleksander, I., & Stonham, T. J. (1979). A guide to pattern recognition using random-access memories. *IEE Proceedings-E Computers and Digital Techniques*, **2**(1), 29–40.

Aleksander, I., Thomas, W., & Bowden, P. (1984). WISARD, a radical new step forward in image recognition. *Sensor Review*, **4**(3), 120–124.

Allinson, N. M., & Johnson, M. (1993). Self-organising *N*-tuple feature maps. *Neural Network World*, (5), 511–530.

Allinson, N. M., & Kolcz, A. (1993). Enhanced *N*-tuple approximators. *Weightless Neural Network Workshop '93*. University of York.

Bledsoe, W., & Browning, I. (1959). Pattern recognition and reading by machine. *IRE Joint Computer Conference* (pp. 225–232).

Cacoullos, T. (1966). Estimation of a multivariate density. *Annals of the Institute of Statistical Mathematics (Tokyo)*, **18**(2), 179–189.

Hand, D. J. (1982). *Kernel discriminant analysis*. Chichester: Research Studies Press (John Wiley).

Härdle, W. (1990). *Applied nonparametric regression*. Cambridge: Cambridge University Press.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359–366.

Johnson, M. (1991). *Self-organising N-tuple feature maps*. PhD thesis, University of York.

Knuth, D. E. (1973). *The art of computer programming*. (Vol. 3). Reading, Mass.: Addison-Wesley.

Kohonen, T. (1984). *Content addressable memories*. Berlin: Springer-Verlag.

Kolcz, A., & Allinson, N. M. (1994). Application of the CMAC input encoding scheme in the *N*-tuple approximation network. *IEE Proceedings-E Computers and Digital Techniques*, **141**(3), 177–183.

Kolcz, A., & Allinson, N. M. (1995). Distance relationships in the *N*-tuple mapping. Electronics Dept/Research Report 1/1995, University of York.

Lapedes, A. S., & Farber, R. (1987). *Nonlinear signal processing using neural networks: Prediction and system modelling*. Technical Report, Los Alamos Laboratory, Los Alamos, New Mexico.

Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, **1**, 281–294.

Nadaraya, E. A. (1964). On estimating regression. *Theory Probab. Applic.*, **15**, 134–7

Park, J., & Sandberg, I. W. (1991). Universal approximation using radial basis function networks. *Neural Computation*, **3**, 246–257.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Science*, **33**, 1065–1076.

Platt, J. (1991). A resource-allocating network for function interpolation. *Neural Computation*, **3**(2), 213–225.

Rosenblatt, M. (1956). Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, **27**, 832–837.

Scott, D. W. (1992). *Multivariate density estimation*. New York: Wiley-Interscience.

Specht, D. F. (1990). Probabilistic neural networks. *Neural Networks*, **3**(1), 109–118.

Specht, D. F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks*, **2**(6), 568–576.

Tattersall, G. D., Foster, S., & Johnston, R. D. (1991). Single-layer lookup perceptrons. *IEE Proceedings-F*, **138**(1), 46–54.

Vieu, P. (1991). Nonparametric regression: optimal local bandwidth choice. *Journal of the Royal Statistical Society B*, **53**(2), 453–464.

Watson, G. S. (1964). Smooth regression analysis. *Sankhya A*, **26**, 359–372.

Widrow, B., & Stearns, S. D. (1985). *Adaptive signal processing*. Englewood Cliffs, NJ: Prentice-Hall.

NOMENCLATURE

$\mathbf{X}, \mathbf{Z}$	D-dimensional vectors of random variables, inputs to the system
$\mathbf{x}, \mathbf{z}$	D-dimensional real vectors, particular instances of $\mathbf{X}$ and $\mathbf{Z}$
Y	random variable corresponding to the system output
y	system output, particular instance of Y
$E(Y/\mathbf{x}), m(\mathbf{x})$	regression function: expected value of the system output, Y, given a particular input vector, $\mathbf{x}$
$\hat{E}(Y/\mathbf{x}), \hat{m}(\mathbf{x})$	estimated value of $E(Y/\mathbf{x})$
$f(\mathbf{x}, y)$	joint probability density function corresponding to the system mapping
$\hat{f}(\mathbf{x}, y)$	estimated value of $f(\mathbf{x}, y)$
$\varphi(x)$	univariate kernel function
$\Phi(\mathbf{x})$	multivariate kernel function
N	size of a tuple
K	number of N-tuples taken
$t_k(\mathbf{x})$	index to the location selected in the k-th tuple memory by a pattern generated for the input, $\mathbf{x}$
$w_k(\mathbf{x})$	the weight value contained in the location pointed by $t_k(\mathbf{x})$
$a_k(\mathbf{x})$	the counter value contained in the location pointed by $t_k(\mathbf{x})$
T	the training-set size
h_T	the smoothing parameter of the kernel function, dependent on the training-set size
$\rho(\mathbf{x}, \mathbf{z})$	the tuple distance function
Q	number of discrete levels, each input variable is quantized to
R	the retina size in bits
H	the Hamming distance between retina patterns
h	normalized Hamming distance
$M_k^i(\mathbf{x})$	an indicator function, equal to one if the tuple selected in the kth memory node by input, $\mathbf{x}$, is identical to the one generated for the ith training sample, $\mathbf{x}^i$ (zero otherwise)
$\|\cdot\|$	Euclidean norm
$\|\cdot\|$	city-block (L_1) norm