N-TUPLE NEURAL NETWORKS

N. M. Allinson and A. R. Kolcz

Department of Electrical Engineering and Electronics, UMIST, Manchester, UK.

The N-Tuple Neural Network (NTNN) is a fast, efficient memory-based neural network capable of performing non-linear function approximation and pattern classification. The random nature of the N-tuple sampling of the input vectors makes precise analysis difficult. Here, the NTNN is considered within a unifying framework of the General Memory Neural Network (GMNN) — a family of networks which include such important types as radial basis function networks. By discussing the NTNN within such a framework, a clearer understanding of its operation and efficient application can be gained. The nature of the intrinsic tuple distances, and the resultant kernel, is also discussed, together with techniques for handling non-binary input patterns. An example of a tuple-based network, which is a simple extension of the conventional NTNN, is shown to yield the best estimate of the underlying regression function, $E(Y|\mathbf{x})$, for a finite training set. Finally, the pattern classification capabilities of the NTNN are considered.

1 Introduction

The origins of the N-tuple neural network date from 1959, when Bledsoe and Browning [1] proposed a pattern classification system that employed random sampling of a binary retina by taking N-bit long ordered samples (i.e., N-tuples) from the retina. These samples form the addresses to a number of memory nodes — with each bit in the sample corresponds to an individual address line. The N-tuple sampling is sensitive to correlations occurring between different regions for a given class of input patterns. Certain patterns will yield regions of the retina where the probability of a particular state of a selected N-tuple will be very high for a pattern class (e.g., predominately 'white' or 'black' if we are considering binary images of textual characters). If a set of exemplar patterns is presented to the retina, each of the N-tuple samples can be thought of as estimating the probability of occurrence of its individual states for each class. A cellular neural network interpretation of N-tuple sampling was provided by Aleksander [2]; and as we attempt to demonstrate in this paper its architecture conforms to what we term as the *General Memory Neural Network* (GMNN). Though the N-tuple neural network is more commonly thought of as a supervised pattern classifier, we will consider first the general problem of approximating a function, f, which exists in a D–dimensional real space, $\mathbb{R}^D$. This function is assumed to be smooth and continuous and that we possess a finite number of sample pairs $\{(\mathbf{x}_i, y_i) : i = 1, 2 \ldots, T\}$. We will further assume that this training data is subject to a noise component, that is $y_i = f(\mathbf{x}_i) + \varepsilon$, where ε is a random error term with zero mean. A variant of the NTNN for function approximation was first proposed by Tattersall *et al* [3] and termed the *Single-Layer-Lookup-Perceptron* (SLLUP). The essential elements of the SLLUP are the same as the basic NTNN except that the nodal memories contain numeric weights. A further extension of the basic NTNN, originally developed by Bledsoe and Bisson [4], records the relative frequencies at which the various nodal memories are addressed during training. The network introduced in Section 4 combines aspects of these two networks and follows directly from the consolidated approach presented in Section 2.

We discuss, in Section 3, some details of the NTNN with particular reference to its mapping between sampled patterns on the retina and the N-tuple distance metric, and the transformation of non-binary element vectors onto the binary retina. The form of the first mapping, which is an approximately exponential function, is the kernel function of the NTNN — though due to the random nature of the sampling, this must be considered in a statistical sense. Finally, a brief note is given on a Bayesian approximation that indicate how these networks can be employed as pattern classifiers.

2 The General Memory Neural Network

Examples of GMNN include *Radial Basis Function* (RBF) [5] networks and the *General Regression Neural Network* (GRNN) [6]. These networks can provide powerful approximation capabilities and have been subject to rigorous analysis. A further class of networks, of which the NTNN is one, have not been treated to such detailed examination. However, these networks (together with others such as the CMAC [7] and the Multi-Resolution Network [8]) are computationally very efficient and better suited to hardware implementation. The essential architectural components of GMNNs are a layer of memory nodes, arranged into a number of blocks, and an addressing element that selects the set of locations participating in the computation of the output response. An extended version of this section is given in [9].

2.1 Canonical Form of the General Memory Neural Network

The GMNN can be defined in terms of the following elements:

- A set of K memory nodes, each possessing a finite number of $|A_k|$ addressable locations.
- An address generator which assigns an address vector
$$A(\mathbf{x}) = [A_1(\mathbf{x}), A_2(\mathbf{x}), \ldots, A_K(\mathbf{x})]$$
to each input point $\mathbf{x}$. The address generated for the kth memory node is denoted by $A_k(\mathbf{x})$.
- The network's output, g, is obtained by combining the contents of selected memory locations, that is
$$[m_1(A_1(\mathbf{x})), m_2(A_2(\mathbf{x})), \ldots, m_k(A_k(\mathbf{x}))] \rightarrow \mathbb{R}, \tag{1}$$
where $m_k(A_k(\mathbf{x}))$ is the content of the memory location selected by the kth memory node by the address generated by $\mathbf{x}$ for that node (this will be identified as simply $m_k(\mathbf{x})$). No specific format is imposed on the nature of the memories other than that the format is uniform for all K nodes.
- A learning procedure exists which permits the network to adjust the nodal memory contents, in response to the training set, so that some error criterion, $\pi(f, g)$, is minimised.

Each node of the network performs a simple vector quantization of the input space into $|A_k|$ cells. For each node, the address generating element can be split into an *index generator* and an *address decoder*. The index generator, I_k, selects a cell for every $\mathbf{x} \in \Omega$ and assigns it a unique index value, $I_k(\mathbf{x}) \in \{1, 2, \ldots, |A_k|\}$; hence the index generator identifies the quantization cell to which the input points belongs. The address decoder uses this generated index value to specify the physical memory address which then selects the relevant node k. Hence, $A_k(\mathbf{x}) = A_k(I_k(\mathbf{x}))$. Therefore, a cell, R_i^k, can be defined as the set of all input points which result in the selection of an address corresponding to the ith index of the kth node.

$$R_i^k = \{\mathbf{x} \in \Omega : I_k(\mathbf{x}) = i\} \tag{2}$$

Each of the cells is closed and bounded as the input space is compact in IR^D. The selection of a cell is given by the following operator or *activation function*

$$S_i^k(\mathbf{x}) = (I_k(\mathbf{x}) = i) = \begin{cases} 1 & \text{if } \mathbf{x} \in R_i^k : i = 1, \ldots, |A_k| \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The quantization of Ω performed by the individual nodes is combined, through the intersection of the K cells being superimposed, to yield a global quantizer. The number of cells $|A|$ is given by the number of all such distinct intersections.

$$|A| = \sum_{i_1=1}^{|A_1|} \sum_{i_2=1}^{|A_2|} \cdots \sum_{i_k=1}^{|A_k|} \prod_{k=1}^{K} (\{\mathbf{x} \in \Omega : I_k(\mathbf{x}) = i_k\} \neq \emptyset) \tag{4}$$

The upper bound being given by $|A|_{\mathrm{max}} = \prod_{k=1}^{K} |A_k|$. The address generation element of the network is distributed across the nodes, so that the general structure of Figure 1a emerges. Alternatively, the address generation can be considered at the global level (Figure 1b). These two variants are equivalent.

The quantization of the input space by the network produces values that are constant over each cell (We are ignoring, for the present, external kernel functions). The value of f assigned to the ith cell is normally expressed as the average value of f over the cell.

$$f : R_i \rightarrow \frac{\int_{R_i} d_f(\mathbf{u}) f_x(\mathbf{u})\, d\mathbf{u}}{\int_{R_i} f_x(\mathbf{u})\, d\mathbf{u}}, \tag{5}$$

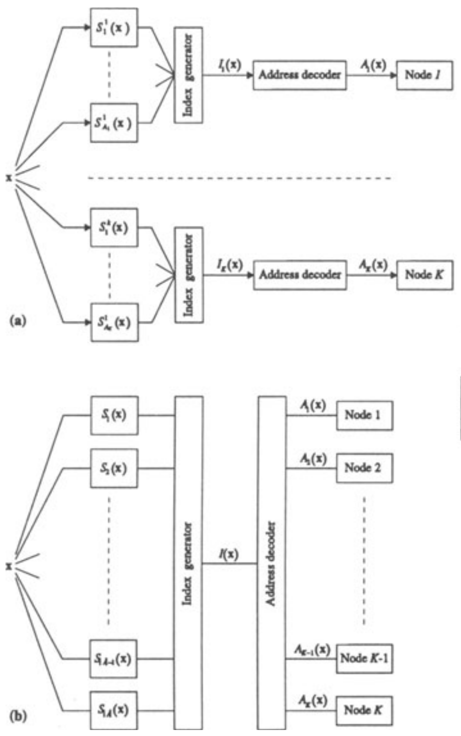
where d_f is given by the squared error function. In most supervised learning schemes, this representation of $f(R_i)$ is estimated inherently through the minimisation of an error function.

For $K = 1$, the GMNN could be simply replaced to a VQ followed by a look-up table. There would need to be at least one input point per quantization cell. The granularity of the quantization needs to be sufficient to meet the degree of approximation performance appropriate for the required task. When there are multiple nodes ($K > 1$), the quantization at the node level can be much coarser which, in turn, increases the probability of input points lying inside a cell. The fine granularity being achieved through the superposition of nodal quantizers. Learning and generalisation are only possible through the use of multiple nodes. Points that are close to each other in the Ω space should share many addresses, and *vice versa*.

2.2 GMNN Distance Metrics

The *address proximity function*, which quantifies the proximity of points in Ω and so the number of generated identical nodal addresses, is given by

$$\kappa : \Omega^2 \rightarrow \{0, 1, \ldots, K\} \quad \kappa(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{K} (A_k(\mathbf{x}) = A_k(\mathbf{y})) = \sum_{k=1}^{K} (I_k(\mathbf{x}) = I_k(\mathbf{y})) \tag{6}$$



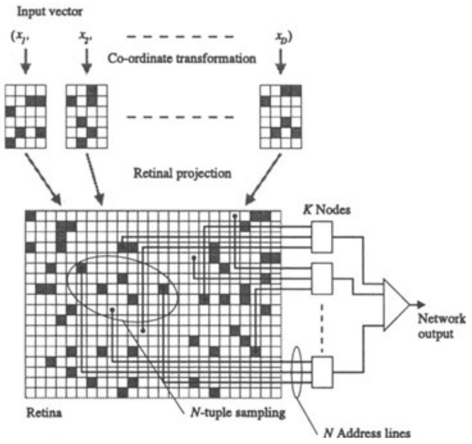


Figure 1 (a) GMNN — address generation considered at the nodal level. (b) GMNN — address generation considered at the global level. These two variants are identical in terms of overall functionality.

Figure 2 General structure of an NTNN.

The *address distance function*, defined as the number of different generated nodal addresses, is given by

$$\rho : \Omega^2 \rightarrow \{0, 1, \ldots, K\} \quad \rho(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{K} (A_k(\mathbf{x}) \neq A_k(\mathbf{y})) = \sum_{k=1}^{K} (I_k(\mathbf{x}) \neq I_k(\mathbf{y})) \quad (7)$$

The *binary nodal incidence function*, which returns '1' if two inputs share a common address at a given network node and '0' otherwise, is defined as

$$M_k(\mathbf{x}, \mathbf{y}) = (I_k(\mathbf{x}) = I_k(\mathbf{y})) = \begin{cases} 1 & \Leftrightarrow \quad I_k(\mathbf{x}) = I_k(\mathbf{y}) \\ 0 & \Leftrightarrow \quad I_k(\mathbf{x}) \neq I_k(\mathbf{y}) \end{cases} \quad (8)$$

From these definitions, several properties directly follow.

$$\begin{array}{lll} \forall(\mathbf{x}, \mathbf{y} \in \Omega) & \rho(\mathbf{x}, \mathbf{x}) = 0 & \rho(\mathbf{x}, \mathbf{y}) = \rho(\mathbf{y}, \mathbf{x}) \\ & \kappa(\mathbf{x}, \mathbf{x}) = K & \kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{y}, \mathbf{x}) \\ & \kappa(\mathbf{x}, \mathbf{y}) = K - \rho(\mathbf{x}, \mathbf{y}) & \end{array} \quad (9)$$

$$\sum_{k=1}^{K} M_k(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x}, \mathbf{y}) \quad (10)$$

2.3 GMNN Error-Based Training

If the address generation elements of the GMNN have been established (based on some *a priori* knowledge about the function to be approximated), then the only element which is modifiable through training is the contents of the nodal memory locations (e.g., the weights). If these locations contain real-valued numbers and the output of the network is formed by summing these numbers, then the response of the GMNN is linear in terms of this weight space. Learning methods based on the minimisation of the square error function are guaranteed to converge under these general circumstances. In the following analysis, we will therefore assume an iterative LMS learning schedule. For a finite training set of T paired samples, the error produced by the network for the jth presentation of the ith training sample is given by

$$y^i - \sum_{k=1}^{K} w_k(\mathbf{x}^i) \tag{11}$$

where $w_k(\mathbf{x}^i)$ is the value of the weight selected by $\mathbf{x}^i$ at the kth node. The participating weights are modified by a value, Δ^i_j, proportional to this error so as to reduce the error.

$$w_k(\mathbf{x}^i) \leftarrow w_k(\mathbf{x}^i) + \Delta^i_j : k = 1, 2, \ldots, K \tag{12}$$

Initially, all weight values are set to zero. As $w_k(\mathbf{x}^i)$ is shared by all points within the neighbourhood $N_k(\mathbf{x}^i)$, this weight updating can affect the network response for points from outside the training set. That is within an input space region given by

$$w_k(\mathbf{x}) \leftarrow w_k(\mathbf{x}) + \Delta^i_j \cdot M_k(\mathbf{x}, \mathbf{x}^i) \tag{13}$$

The output of the network after the training is complete, for an arbitrary $\mathbf{x} \in \Omega$, will depend on all training samples lying within the neighbourhood of $\mathbf{x}$.

$$g(\mathbf{x}) = \sum_{k=1}^{K} w_k(\mathbf{x}) = \sum_{k=1}^{K}\sum_{i=1}^{T}\left(M_k(\mathbf{x}, \mathbf{x}^i)\sum_{j=1}^{T_i}\Delta^i_j\right) \tag{14}$$

Rearranging this expression and using the identity (10), yields

$$g(\mathbf{x}) = \sum_{i=1}^{T}\sum_{j=1}^{T_i}\Delta^i_j\sum_{k=1}^{K} M_k(\mathbf{x}, \mathbf{x}^i) = \sum_{i=1}^{T}\Delta^i \cdot \kappa(\mathbf{x}, \mathbf{x}^i), \text{ where } \sum_{j=1}^{T_i}\Delta^i_j = \Delta^i. \tag{15}$$

We can compare this result with the response of a trained RBF network.

$$g(\mathbf{x}) = \sum_{i=1}^{T} w^i \cdot \kappa(\mathbf{x}, \mathbf{x}^i) \tag{16}$$

For the normalised RBF network, the response is given by

$$g(\mathbf{x}) = \frac{\sum_{i=1}^{T} w^i \cdot \kappa(\mathbf{x}, \mathbf{x}^i)}{\sum_{i=1}^{T} \kappa(\mathbf{x}, \mathbf{x}^i)} \tag{17}$$

and for the GRNN (where the training set response values replace the weight values).

$$g(\mathbf{x}) = \frac{\sum_{i=1}^{T} y^i \cdot \kappa(\mathbf{x}, \mathbf{x}^i)}{\sum_{i=1}^{T} \kappa(\mathbf{x}, \mathbf{x}^i)} \tag{18}$$

Though there are obvious similarities, we can further extend the functionality of the GMNN by incorporating into each nodal memory an integer counter. This nodal counter is incremented whenever the node is addressed, that is $c_k(\mathbf{x}^i) \leftarrow c_k(\mathbf{x}^i) + 1$ (Initially, all counter values are set to zero). Now the response of the GMNN is given by

$$g(\mathbf{x}) = \frac{\sum_{i=1}^{T} \Delta^i \cdot \kappa(\mathbf{x}, \mathbf{x}^i)}{\sum_{i=1}^{T} \kappa(\mathbf{x}, \mathbf{x}^i)} \tag{19}$$

The trained GMNN is equivalent to a set of T units, each centred at one of the training samples, $\mathbf{x}^t$, and possessing height Δ^t and kernel weighting function $\kappa(\cdot, \mathbf{x}^t)$. To complete the equivalence of the GMNN and the GRNN, κ must satisfy the general conditions imposed on kernel functions [10].

2.4 Introduction of External Kernels

The network output is given by the sum of weights corresponding to the selected locations, but the output remains constant over each quantization cell — regardless of the relative position of the input point inside a cell. The network mapping thus becomes discontinuous at the cell boundaries. A solution would be to emphasise weights that correspond to quantization cells that are closer to the current excitation, $\mathbf{x}$, than others. This distance can be defined in terms of the distance between $\mathbf{x}$ and the centroid of R_i^k (where $i = I_k(\mathbf{x})$).

$$d(\mathbf{x}, R_i^k) = d(\mathbf{x}, c_i^k) \tag{20}$$

A smooth, continuous and monotonically decreasing kernel function is then introduced to weight the contributions of the respective nodes to the values of $d(\mathbf{x}, R_i^k(\mathbf{x}))$, where $R_i^k(\mathbf{x})$ is the cell selected by $\mathbf{x}$ for the kth node. The network output now becomes

$$g(\mathbf{x}) = \sum_{k=1}^{K} \sum_{i=1}^{|A_k|} w_i^k \cdot \varphi_i^k(d(\mathbf{x}, R_i^k(\mathbf{x}))) \cdot M_k(c_i^k, \mathbf{x}) \tag{21}$$

A set of $K \cdot |A_k|$ kernel or basis functions can be defined, with centres given by the centroid set $\{c_i^k\}$ and where each kernel has its support truncated to its corresponding quantization region. The network mapping can be expressed as

$$g(\mathbf{x}) = \sum_{k=1}^{K} \sum_{i=1}^{|A_k|} w_i^k \cdot \varphi_i^k(\mathbf{x}) \tag{22}$$

or in its normalised form as

$$g(\mathbf{x}) = \frac{\sum_{k=1}^{K} \sum_{i=1}^{|A_k|} w_i^k \cdot \varphi_i^k(\mathbf{x})}{\sum_{k=1}^{K} \sum_{i=1}^{|A_k|} \varphi_i^k(\mathbf{x})} \tag{23}$$

$\varphi_i^k(\mathbf{x})$ is the kernel function associated with the ith quantization cell of the kth node and truncated to zero at its cell boundaries. Gaussian kernels provide an approximation to this last condition, though B-spline kernels [11] can lead to the total absence of cell discontinuities. The introduction of external weighting kernels is the final step in the GMNN architecture.

3 The N-Tuple Neural Network

The general form of the NTNN was described in the introduction and is shown in Figure 2. The following two sections consider the mapping functions inherent to this network. Namely:

- Conversion of the input vector into the binary format needed for the retina
- Sampling the retina by taking N bit values at a time to the address of one of the K memory nodes.

There is some choice in what form the first of these mappings may take depending on the application, but the retinal N-tuple sampling is common to all NTNNs. Figure 3a indicates how the threshold decision planes of the individual elements of a tuple delineate the input space into discrete regions and why the Hamming distance between tuple values is the obvious choice for a distance metric. Further details of the N-tuple distance metric and input encoding are given in [12, 13].

3.1 Retinal Sampling

The relationship between the number of different addresses generated for two arbitrary inputs, $\mathbf{x}$ and $\mathbf{y}$, and the Hamming distance $H(\mathbf{x},\mathbf{y})$ (i.e., the number of bits for which $\mathbf{x}$ and $\mathbf{y}$ differ) is important as it reveals the nature of the distance metric necessary when a NTNN is used for pattern classification and the form of the kernel for the approximation-NTNN. This relationship can only be expressed in terms of an expectation due the random nature of the sampling. For sampling without repetitions, the expected value of $\rho_{\mathrm{NTNN}}(\mathbf{x},\mathbf{y}) = \rho_{\mathrm{NTNN}}(H)$ is given by

$$E(\rho_{\mathrm{NTNN}}(H)) = K\left(1 - \left(1 - \frac{1}{K}\right)^H\right) \tag{24}$$

For small values of H, this can be simplified to

$$E(\rho_{\mathrm{NTNN}}(H)) \approx K\left(1 - \exp\left(-\frac{H}{K}\right)\right) \tag{25}$$

For sampling with repetitions, the expected value is

$$E(\rho_{\mathrm{NTNN}}(H)) = K\left(1 - \exp\left(N \cdot \left(h - \frac{h^2}{2} + \frac{h^3}{3} - \cdots\right)\right)\right) \tag{26}$$

where h is the normalised Hamming distance ($= H/R$). Again, this can be simplified for small values of h, to yield

$$E(\rho_{\mathrm{NTNN}}(H)) \approx K(1 - \exp(-N \cdot h)) \tag{27}$$

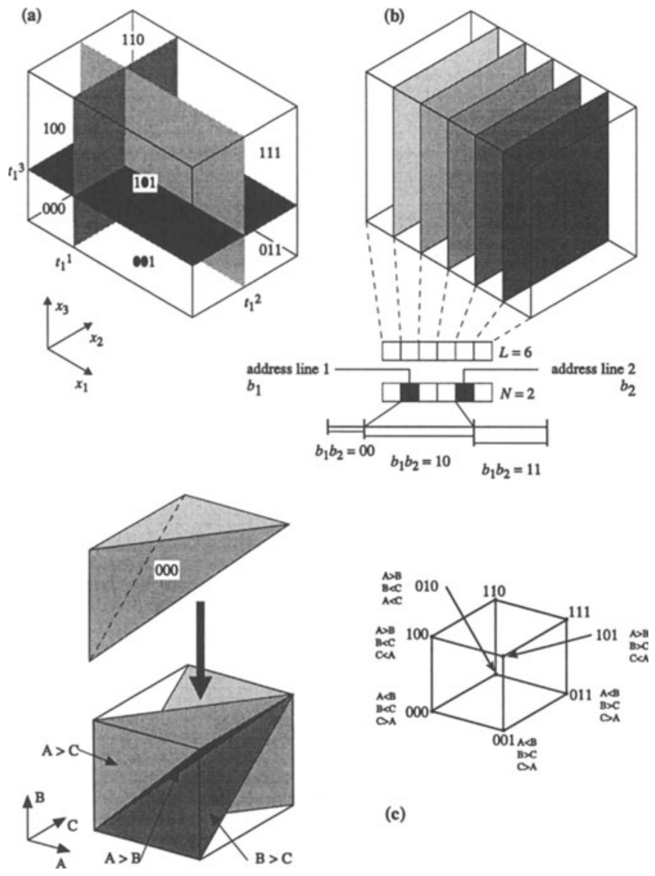


Figure 3 (a) The delineation of input space by the hard decision planes of each tuple element's threshold. Each region is marked by its specific binary state of the 3-tuple, t_1. (b) The thermometer coding inherent in N-tuple sampling. The variable, x_1, is uniformly quantized into six discrete regions ($L = 6$). The indicated 2-tuple partitions this interval into three unequal quantization regions, with the binary state of the 2-tuple indicated. (c) The delineation of the 3-dimensional input space into tetrahedral regions through the use of a ranking code. The binary space representation of the input space is also shown.

There is little difference in the general form of these two sampling methods, though there may be crucial differences in performance for specific tasks. The distance function depends exponentially on the ratio of the Hamming distance, H, between patterns to the retinal size, R. The rate of decrease is proportional to the tuple size.

3.2 Input Encoding

There is a direct and monotonic dependence of ρ_{NTNN} on the Hamming distance in the binary space of the network's retina. For binary patterns, the N-tuple sampling provides the desired mapping between the input and output addresses. For non-binary input patterns, the situation is not so clear. One obvious solution is to use a thermometer or bar-chart code, where one bit is associated to every level of an input integer. This creates a linear array of 2^n bits for an n-bit long integer. This can produce very large retinas if the input dimensionality and quantization level are large. The use of the natural binary code or Gray code is not feasible. Though these are compact codes, there is no monotonic relationship between input and pattern distances. The concatenation of several Gray codes [3] offers an improvement over a limited region and enhances the dynamic range over the binary and straight Gray code. The exponential dependence of the ρ_{NTNN} on the Hamming distance means that strict proportionality is not required but monotonicity is required within an *active* region of Hamming distances.

The potential of CMAC encoding, and further aspects of input coding methods, are discussed in Kolcz and Allinson [14]. Improvements in the input mapping, which in turn produce a more isotropic kernel, are given in Kolcz and Allinson [15], where rotation and hyper-sphere codings are described. Two further techniques will be briefly introduced here in order to indicate the wide range of possible sampling and coding schemes. Figure 3b shows one input variable, x_1, which is uniformly quantized to six levels and this is sampled by the indicated 2-tuple. The corresponding states of the resultant tuple for the three resulting sub-intervals indicate that thermometer encoding can be inherent in tuple sampling. This concept can be extended to the multivariate case. If the input space, Ω, is a D-dimensional hypercube and each memory node distributes its N address lines among these dimensions, then the space is effectively quantized into $\prod_{d=1}^{D}(N_d+1)$ hyper-rectangular cells. This assumes random sampling, such that there are N_d address lines per input dimensions (where $N_d = N/D$). The placement of tuples can be very flexible (i.e., uniform quantization is not essential) and the sampling process can take into account the density of training points within the input space.

In rank-order coding, the non-binary N-tuple is transformed to a tuple of ranked values (e.g., $\langle 20, 63, 40, 84, 122, 38\rangle$ becomes, in ascending order, the ranked tuple $\langle 0, 3, 2, 4, 5, 1\rangle$). Each possible ordering is assigned a unique consecutive ranking number, which is converted to binary format and then used as the retinal input. Rank-order coding produces an equal significance code. The use of these relationships is equivalent to delineating the input space into hyper-tetrahedrons rather than the usual hyper-rectangles (Figure 3c).

4 N-tuple Regression Network

The framework for GMNN proposed earlier together with the derivation the tuple distance metric are employed here in the development of a modified NTNN which operates as a non-parametric regression estimator. The formal derivation of this network and that the N-tuple kernel is a valid one for estimating the underlying probability function is given in Kolcz and Allinson [16]. The purpose of this section is to show the relative simplicity of this network compared with other implementations.

During the training phase, the network is presented with T data pairs, $(\mathbf{x}^i, y^i)$, where $\mathbf{x}^i$ is the D-dimensional input vector and y^i is the corresponding scalar output of the system under consideration. The input vector is represented by a unique selection of the K tuple addresses with their associated weight and counter values.

$$\mathbf{x} \rightarrow \begin{cases} \{t_1(\mathbf{x}), t_2(\mathbf{x}), \ldots, t_K(\mathbf{x})\} \\ \{w_1(\mathbf{x}), w_2(\mathbf{x}), \ldots, w_K(\mathbf{x})\} \\ \{a_1(\mathbf{x}), a_2(\mathbf{x}), \ldots, a_K(\mathbf{x})\} \end{cases} \tag{28}$$

During training, each addressed tuple location is updated according to

$$w_k(\mathbf{x}^i) \leftarrow w_k(\mathbf{x}^i) + y^i \text{ and } a_k(\mathbf{x}^i) \leftarrow a_k(\mathbf{x}^i) + 1 \tag{29}$$
$$\text{for } i = 1, 2, \ldots, T \text{ and } k = 1, 2, \ldots, K$$

Initially all weight and counter values are set to zero. After training, the network output, $\hat{y}(\mathbf{x})$, is obtained from

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{x})}{\sum_{k=1}^{K} a_k(\mathbf{x})} \tag{30}$$

An additional condition is where all addressed locations are zero. In this case, the output is set to zero.

$$\sum_{k=1}^{K} a_k(\mathbf{x}) = 0 \rightarrow \hat{y}(\mathbf{x}) = 0 \tag{31}$$

Figure 4 shows the modifications needed to a conventional NTNN to form the N-

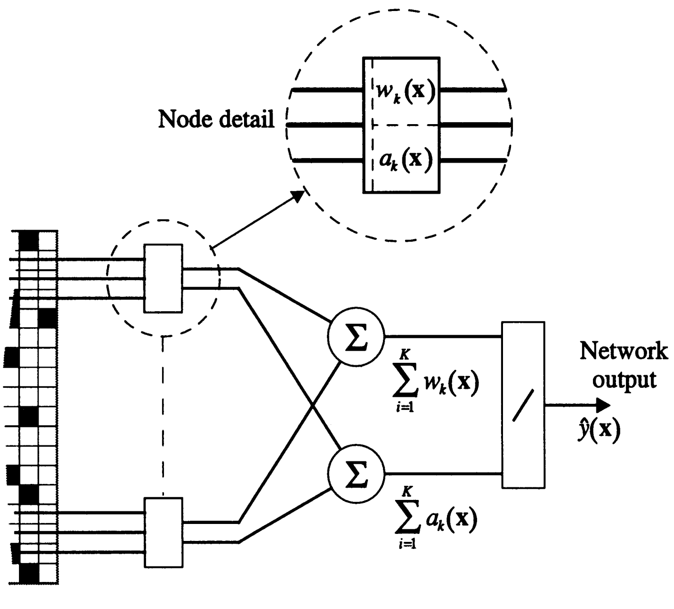


Figure 4 Modifications to the nodes and output elements of the NTNN to yield the N-tuple regression network.

tuple regression network. By considering the tuple distances between inputs, as defined in terms of the number of different tuple addresses generated, then (30) can be extended to

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k(\mathbf{x})}{\sum_{k=1}^{K} a_k(\mathbf{x})} = \frac{\sum_{i=1}^{T} y^i \cdot \left(1 - \frac{\rho(\mathbf{x},\mathbf{x}^i)}{K}\right)}{\sum_{i=1}^{T} \left(1 - \frac{\rho(\mathbf{x},\mathbf{x}^i)}{K}\right)} \tag{32}$$

This suggests that the network output is an approximate solution of the generalised regression function, $E(Y|\mathbf{x})$, provided that the bracketed term in (32) is a valid kernel function. This function is continuous, symmetrical, non-negative and possesses finite support. These are all necessary conditions. A close approximation (based on the exponential approximation of tuple distances) is also representable as a product of univariate kernel functions. Taken together these provide sufficient conditions for a valid kernel function [17]. A wide ranging set of experiments on chaotic time-series prediction and non-linear system modelling has been conducted [16], which confirm the successful operation of this network. A major advantage of the NTNN implementation over other approaches is its fast, and fixed, speed of operation. Each recall operation involves addressing a fixed number of locations. There is no need for preprocessing large data sets, through data clustering, as is often the case for RBF networks [18].

5 Pattern Classification

So far we have restricted our considerations to the approximation properties of the NTNN, but the other major application — namely, pattern classification — can be discussed within this common framework. The training phase of a supervised network provides estimates of the conditional probabilities of individual pattern classes. The class membership probabilities can be formulated through the Bayes relationship, i.e.,

$$P(\mathbf{x} \in c) = \frac{P(\mathbf{x}|c)P(c)}{P(\mathbf{x})} \tag{33}$$

where c is the class label for a particular class $\{c = 1, 2, \ldots, C\}$. The modified NTNN discussed in Section 4 can be reformulated in terms of this classification. The network through training approximates C indicator functions, which denote membership to an individual class.

$$I_c(\mathbf{x}) = \begin{cases} 1 & \text{if } x \subset c \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

Modifying (32), the indicator functions can be approximated, after training, by

$$I_c(\mathbf{x}) = \frac{\sum_{i=1}^{T} (\mathbf{x}^i \subset c)(1 - \rho(\mathbf{x},\mathbf{x}^i)/K)}{\sum_{i=1}^{T} (1 - \rho(\mathbf{x},\mathbf{x}^i)/K)} = \frac{\sum_{k=1}^{K} w_k^c}{\sum_{k=1}^{K} a_k} \equiv P(c)\frac{\sum_{k=1}^{K} P(t_k|c)}{\sum_{k=1}^{K} P(t_k)} \tag{35}$$

This relationship gives the ratio of the cumulative summation of all training points belonging to a class c, which have an N-tuple distance at $0, 1, \ldots, (K-1)$ from $\mathbf{x}$ to a similar cumulative summation for all training points. The decision surfaces present in the K-dimensional weight space are described by $\sum_{k=1}^{K} w_k = \text{const}$, and the winning class is given by $c_{\text{winner}} = \max_{c=1,2,\ldots,C} \sum_{k=1}^{K} w_k^c$.

6 Conclusions

The unifying approach proposed for a wide class of memory-based neural networks means that practical, but poorly understood, networks (such as the NTNN) can be considered in direct comparison with networks (such as RBF networks) that possess a much firmer theoretical foundation. The random sampling inherent in the N-tuple approach makes detailed analysis difficult so this link is all the more important. The pragmatic advantages of NTNNs has been demonstrated in the regression network described above, where large data-sets can be accommodated with fixed computational overheads. The possible range of input sampling and encoding strategies has been illustrated, but by no means exhaustively. There is still a need to seek other strategies that will provide optimum kernel functions for specified recognition or approximation tasks. The power and flexibility of Bledsoe and Browning's original concept has not, as yet, been fully exploited.

REFERENCES

[1] Bledsoe W W and Browning I, *Pattern recognition and reading by machine*, IRE Joint Computer Conference, 1959, 225–232.

[2] Aleksander I, *Fused adative circuit which learns by example*, Electronics Letters, 1965, 1, 173–174.

[3] Tattersall G D, Foster S and Johnston R D, *Single-layer lookup perceptrons*, IEE Proceedings — F: Radar and Signal Processing, 1991, 138, 46–54.

[4] Bledsoe W W and Bisson C L, *Improved memory matrices for the N-tuple pattern recognition method*, IRE Transactions on Electronic Computers, 1962, 11, 414–415.

[5] Broomhead D S and Lowe D, *Multivariable functional interpolation and adaptive networks*, Complex Systems, 1988, 2, 321–355.

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

[7] Albus J S, *A new approach to manipulator control: the cerebellar model articulation controller (CMAC)*, Journal of Dynamic Systems, Measurement and Control, 1975, 97, 220–227.

[8] Moody J, *Fast learning in multi-resolution hierarchies*, in Advances in Neural Information Processing 1 (Touretzky D S, ed.), 1989, Morgan Kaufmann: San Mateo, CA, 29–39.

[9] Kolcz A and Allinson N M, *General Memory Neural Network — extending the properties of basis networks to RAM-based architectures*, 1995 IEEE International Conference on Neural Networks, Perth, Western Australia.

[10] Park J and Sandberg, I W, *Universal approximation using radial basis function networks*, Neural Computation, 1991, 3, 246–257.

[11] Kavli T, *ASMOD - an algorithm for adaptive modelling of observational data*, International Journal of Control, 1993, 58, 947–967.

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

[13] Kolcz A and Allinson N M, *Enhanced N-tuple approximators*, Proceedings of Weightless Neural Network Workshop 93, 1993, 38–45.

[14] Allinson N M and Kolcz A, *The theory and practice of N-tuple neural networks*, in Neural Networks (Taylor J G, ed.), 1995, Alfred Waller, 53–70.

[15] Kolcz A and Allinson N M, *Euclidean mapping in an N-tuple approximation network*, Sixth IEEE Digital Signal Processing Workshop, 1994, 285–289.

[16] Kolcz A and Allinson N M, *N-tuple regression network*, Neural Networks vol 9 No.5 pp855-870.

[17] Parzen E, *On estimation of a probability density function and mode*, Annals of Mathematical Statistics, 1962, 33, 1065–1076.

[18] Moody J and Darken C J, *Fast learning in networks of locally-tuned processing units*, Neural Computation, 1989, 1, 281–294.