2-1997

Inductive neural logic network and the SCM algorithm

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

Loo-Nin TEOW

ELSEVIER

Neurocomputing 14 (1997) 157–176

NEUROCOMPUTING

Inductive neural logic network and the SCM algorithm

Ah-Hwee Tan *, Loo-Nin Teow

Real World Computing Partnership (RWCP), Neuro ISS Laboratory, Institute of Systems Science, National University of Singapore, Heng Mui Keng Terrace, Kent Ridge, Singapore 119597

Received 2 May 1995; accepted 31 January 1996

Abstract

Neural Logic Network (NLN) is a class of neural network models that performs both pattern processing and logical inferencing. This article presents a procedure for NLN to learn multi-dimensional mapping of both binary and analog data. The procedure, known as the Supervised Clustering and Matching (SCM) algorithm, provides a means of inferring inductive knowledge from databases. In contrast to gradient descent error correction methods, pattern mapping is learned by an inductive NLN using fast and incremental clustering of input and output patterns. In addition, learning/encoding only takes place when both the input and output match criteria are satisfied in a template matching process. To handle sparse and/or noisy data sets, we also present a weighted voting scheme whereby distributed cluster activities combine to produce a final output. The performance of the SCM algorithm, compared with alternative systems, is illustrated on three benchmark problems: (1) mushroom classification, (2) sonar return signal recognition, and (3) sunspot time series prediction.

Keywords: Supervised learning; Incremental clustering; Template matching

1. Introduction

Neural Logic Network (NLN) is a class of neural network models that performs both pattern processing and logical inferencing [19–21]. Although defined in terms of nodes and links, an NLN functions like a logical system. By extending Boolean logic to 3-valued logic and soft logic, including probabilistic logic and fuzzy logic, a neural logic network is a powerful model for simulating human logical reasoning. As a neural

* Corresponding author. Email: ahhwee@iss.nus.sg.

PII S0925-2312(96)00032-X

network model, a neural logic network performs massively parallel pattern learning and recognition. A specialized 3-layer architecture enables pattern matching from a set of input patterns to a set of output patterns. We call Neural Logic Network with such a architecture and purpose, *inductive Neural Logic Network*. Given a *consistent* mapping problem such that no single input pattern is associated with two distinct output patterns, a construction algorithm determines the required network topology to perform the given mapping [14,21].

Although the construction algorithm proves the existence of an NLN for each and every consistent mapping problem, it suffers from two limitations. First, as an intermediate node is assigned to learn the mapping between a pair of input and output patterns, the network size scales up linearly as the number of pattern pairs increases. Second, the weight assignment process is based on exact match firing condition. The constructed network thus cannot tolerate error in the input patterns or perform generalization. Although algorithms, such as *feature enhancement* and *fine tuning* [11,14,16,22], solve the latter problem to a certain extent, it is unclear how much generalization a network can obtain by retaining the exact match firing condition during performance.

This article introduces a learning algorithm that incorporates a clustering mechanism into the construction process. The Supervised Clustering and Matching (SCM) algorithm [17,18] is adapted from the dynamics of a supervised Adaptive Resonance Theory (ART) [1,2] model termed Adaptive Resonance Associative Map (ARAM) [15]. Instead of encoding a pair of input and output patterns, each hidden node now encodes a pair of input and output templates of a cluster of input and output patterns respectively. Whereas typical clustering systems, such as ART and LVQ [8], perform unsupervised categorization of input patterns, SCM makes use of a template matching process to supervise the input clustering. Specifically, learning/encoding only takes place when both the input and output patterns match *well* with their respective weight templates, according to some similarity measures. SCM is similar to another supervised learning algorithm for NLN, known as the Supervised Incremental Clustering Algorithm (SICA) [9,10], that is also based on supervised clustering. However, whereas SICA considers output in the form of labels or classes, SCM represents output in the form of continuous patterns. This enables inductive NLN to perform a wider range of memory tasks including both pattern classification and function approximation.

Based on the SCM learning mechanism, a set of system equations is derived for each sub-class of NLN, namely 3-valued NLN, Boolean NLN, and fuzzy NLN. The SCM algorithm is evaluated on three well-known benchmark problems obtained from a public domain machine learning directory [12], namely (1) mushroom classification, (2) sonar return recognition, and (3) sunspot prediction. The mushroom problem is a pattern classification problem based on binary attributes. We apply both Boolean NLN and 3-valued NLN to the mushroom problem and find that 3-valued NLN forms a more compact internal representation. We apply fuzzy NLN to the second and third benchmark problems, that are based on real-valued features. Whereas sonar return recognition is a classification problem, sunspot time series prediction is a function approximation problem. In all benchmark simulations, the results indicate that besides the advantage of fast incremental learning, the SCM algorithm performs competently, in terms of network size and test set generalization, compared with alternative learning systems, including K

nearest neighbor system, fuzzy ARTMAP [3], fuzzy ARAM [15], back-propagation network [24], and threshold Autoregressive model [23].

The rest of this article is organized as follows. Section 2 provides an overview of Neural Logic Network and motivates the SCM learning algorithm. Section 3 presents the SCM algorithm and the equations for 3-valued NLN and Fuzzy NLN. Section 4 compares the performance of the SCM algorithm with existing learning methods based on the three benchmark problems. Concluding remarks and future extensions are discussed in Section 5.

2. Neural logic network

An *abstract* neural logic network (NLN) [19,20] is a mathematical system denoted by

$$< \mathcal{N}, \mathcal{E}, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{A}, \psi_1, \psi_2, \mathcal{F} > ,$$

where

$\mathcal{N}$ is the set of nodes of directed graphs,
$\mathcal{E}$ is the chosen set of links,
$\mathcal{I}$ is the chosen set of input nodes,
$\mathcal{O}$ is the chosen set of output nodes,
$\mathcal{R}$ is the chosen ring,
$\mathcal{A}$ is the chosen subset of $\mathcal{R}$ whose elements are called the truth values,
ψ_1 is a mapping from $\mathcal{E}$ to $\mathcal{R}$ which assigns weights to links,
ψ_2 is the mapping from $(\mathcal{N} - \mathcal{I})$ to $\mathcal{A}$ which assigns truth values to non-input nodes, and $\mathcal{F}$ is the mapping from $\mathcal{R}$ to $\mathcal{A}$ called the threshold function.

By an appropriate choice of the ring structure $\mathcal{R}$, the truth value domain $\mathcal{A}$, and the threshold function $\mathcal{F}$, different sub-classes of NLN can be obtained. Three important subclasses are Boolean NLN, 3-valued NLN, and fuzzy NLN.

One unique feature of NLN is the use of ordered-pair values to represent both

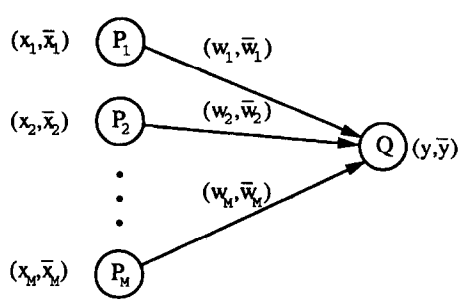


Fig. 1. A single layer 3-valued Neural Logic Network.

positive and negative truth values. Specifically, for 3-valued NLN, we take $\mathscr{A} = \{(1,0),(0,1),(0,0)\}$ as the chosen set of truth values, where (1,0) denotes "True", (0,1) denotes "False", and (0,0) denotes "Unknown". Given a node Q with a set of incoming links from nodes $P_1, P_2, \ldots, P_M$, let $(w_i, \bar{w}_i)$ be the weight on the edge from P_i to Q and $(x_i, \bar{x}_i)$ be the truth value of P_i (Fig. 1). The truth value of node Q, $(y, \bar{y})$ is computed by

$$(y, \bar{y}) = \mathscr{F}(t, \bar{t}), \tag{1}$$

where the positive truth t is given by

$$t = \sum_i w_i x_i, \tag{2}$$

the negative truth $\bar{t}$ is given by

$$\bar{t} = \sum_i \bar{w}_i \bar{x}_i \tag{3}$$

and the double step function $\mathscr{F}: R^2 \to A$ is defined by

$$\mathscr{F}(t, \bar{t}) = \begin{cases} (1,0) & \text{if } T \geq 1 \\ (0,1) & \text{if } T \leq -1 \\ (0,0) & \text{otherwise,} \end{cases} \tag{4}$$

where the net truth T is computed by

$$T = t - \bar{t}. \tag{5}$$

3-valued logic is more complicated and potentially more powerful than Boolean logic. Assuming two 3-valued inputs, there are nine different input combinations and thus nine output values for each 3-valued logical operator. Since each output can take one of the three values and a different assignment of values to the nine outputs results in a distinct operator, there are altogether 19,683 (3^9) 3-valued logical operators. 3-valued NLN has very rich logical properties in the sense that each of the 19,683 different 3-valued logical operators can be realized by a single simple 3-valued NLN, by suitable substitutions of weight values [19]. Some of the more basic operators, such as 3-valued AND, OR, and XOR, are natural generalizations of their corresponding Boolean counterparts.

Although NLN has very strong logical properties, in this paper, we shall focus on another interesting task performed by NLN, i.e., pattern matching and classification. The cornerstone of the pattern matching properties is an existence theorem stated below.

Theorem (3-valued pattern matching). *Let* $\mathscr{A} = \{(1,0),(0,1),(0,0)\}$. *Given a mapping* $\theta: \mathscr{A}^M \to \mathscr{A}^N$, *where M and N are positive integers, that matches a set of input patterns to a set of output patterns, if the mapping is consistent such that no single input pattern maps to two different output patterns, there exists a 3-valued neural logic network which induces the mapping* θ.

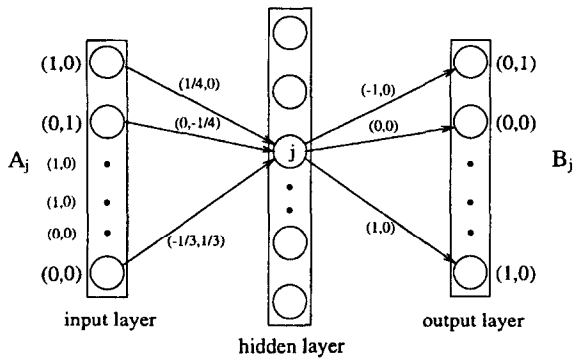


Fig. 2. Using the construction algorithm, each hidden node j in a 3-layer feedforward inductive Neural Logic Network encodes a pattern pair $(\mathbf{A}_j,\mathbf{B}_j)$.

The detailed proof of the 3-valued pattern matching theorem has been presented elsewhere [21]. To make the article self-contained, a summary of the construction algorithm, which is central to the proof, is presented below.

2.1. Construction algorithm

Given a set of input and output pattern pairs $\{(\mathbf{A}_1,\mathbf{B}_1),(\mathbf{A}_2,\mathbf{B}_2),\ldots,(\mathbf{A}_S,\mathbf{B}_S)\}$, the algorithm constructs a 3-layer feedforward neural logic network that maps each $\mathbf{A}_j$ to its corresponding $\mathbf{B}_j$.

1. Let M be the dimension of the input patterns $\mathbf{A}$ and N be the dimension of the output patterns $\mathbf{B}$. Construct a 3-layer feedforward network with M nodes in the input layer, S nodes in the hidden layer, and N nodes in the output layer. Add links such that the network is fully connected between consecutive layers (Fig. 2).
2. Let $\mathbf{w}^a$ denote the weight matrix between the input and hidden layers. Let $\mathbf{w}^b$ denote the weight matrix between the hidden and output layers. For each input and output pair $(\mathbf{A}_j,\mathbf{B}_j)$, where

$$\mathbf{A}_j = \left(\left(a_1^j,\bar{a}_1^j\right),\left(a_2^j,\bar{a}_2^j\right),\ldots,\left(a_M^j,\bar{a}_M^j\right)\right) \tag{6}$$

and

$$\mathbf{B}_j = \left(\left(b_1^j,\bar{b}_1^j\right),\left(b_2^j,\bar{b}_2^j\right),\ldots,\left(b_N^j,\bar{b}_N^j\right)\right), \tag{7}$$

the weight vectors

$$\mathbf{w}_j^a = \left(\left(w_{j1}^a,\bar{w}_{j1}^a\right),\left(w_{j2}^a,\bar{w}_{j2}^a\right),\ldots,\left(w_{jM}^a,\bar{w}_{jM}^a\right)\right), \tag{8}$$

and

$$\mathbf{w}_j^b = \left(\left(w_{j1}^b,\bar{w}_{j1}^b\right),\left(w_{j2}^b,\bar{w}_{j2}^b\right),\ldots,\left(w_{jN}^b,\bar{w}_{jN}^b\right)\right) \tag{9}$$

associated with the hidden node j are determined by

$$\left(w_{ji}^{a},\bar{w}_{ji}^{a}\right)=\begin{cases}\left(\dfrac{1}{c},0\right) & \text{if } \left(a_i^j,\bar{a}_i^j\right)=(1,0)\\ \left(0,-\dfrac{1}{c}\right) & \text{if } \left(a_i^j,\bar{a}_i^j\right)=(0,1)\\ \left(-\dfrac{1}{d+1},\dfrac{1}{d+1}\right) & \text{if } \left(a_i^j,\bar{a}_i^j\right)=(0,0)\end{cases} \tag{10}$$

and

$$\left(w_{ji}^{b},\bar{w}_{ji}^{b}\right)=\begin{cases}(1,0) & \text{if } \left(b_i^j,\bar{b}_i^j\right)=(1,0)\\ (-1,0) & \text{if } \left(b_i^j,\bar{b}_i^j\right)=(0,1)\\ (0,0) & \text{if } \left(b_i^j,\bar{b}_i^j\right)=(0,0),\end{cases} \tag{11}$$

where $c=\sum_i(a_i^j+\bar{a}_i^j)$ and $d=M-c$.

It can be verified that the constructed network maps each and every input pattern $\mathbf{A}_j$ to its corresponding output pattern $\mathbf{B}_j$. When the input pattern $\mathbf{A}_j$ of a pattern pair $(\mathbf{A}_j,\mathbf{B}_j)$ is presented, only hidden node j will be activated and read out the output pattern $\mathbf{B}_j$. Although the algorithm proves the existence of an NLN to implement any given mapping defined by a finite set of input output pairs, one hidden node is needed to encode each pattern pair. In other words, the network size scales up linearly as the number of examples increases. To solve this problem, we propose to use a clustering or compression mechanism to reduce the number of hidden nodes for a more compact internal representation.

3. Learning by supervised clustering and matching

The SCM algorithm replaces the exact match firing condition by a competitive activation process among hidden nodes. Instead of encoding a pair of input and output patterns, each hidden node now encodes a pair of input and output templates of a *cluster* of input and output patterns respectively. The hidden nodes are thus called the *cluster* nodes (Fig. 3). Given a pair of input and output patterns, the algorithm first identifies a cluster node based on the input pattern. It then checks if the input and output templates of the cluster node are *close* enough to the input and output patterns respectively, based on a similarity measure. If both of the matches satisfy their respective *vigilance* criteria, the cluster node *learns* the given input and output patterns by tuning its template patterns. Otherwise, the system resets the cluster node and repeats to select another node. If no existing cluster node satisfies the above condition, a new cluster node is recruited to encode the given pattern pairs. With a stricter vigilance criterion (that corresponds to a higher vigilance parameter values), new cluster nodes are more likely to be created. For most classification applications, a low input vigilance value and a high output vigilance value provide maximal generalization with a minimal number of cluster

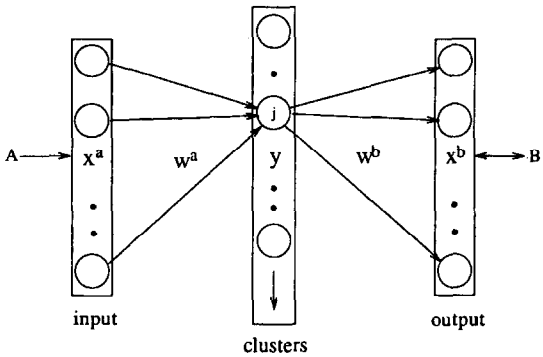


Fig. 3. Using the SCM algorithm, each hidden/cluster node j encodes a set of input and output pattern pairs. A new cluster is created only when a new pattern does not fit into any existing cluster.

nodes. In applications where highly similar input patterns often correspond to distinct output patterns, a high input vigilance value is needed to restrict the size (generalization range) of cluster nodes.

In the following sections, we provide the SCM equations of 3-valued NLN and fuzzy NLN. The equations of Boolean NLN are omitted as they can be derived from that of 3-valued NLN by dropping the negative terms.

3.1. 3-valued neural logic network

Input and output patterns: Let $(a_i,\bar{a}_i)$ denote the i-th attribute value of the input vector **A** and let $(b_k,\bar{b}_k)$ denote the k-th attribute value of the output vector **B**.

Truth value vectors: Let

$$\mathbf{x}^a = \left(\left(x_1^a,\bar{x}_1^a\right),\left(x_2^a,\bar{x}_2^a\right),\ldots,\left(x_M^a,\bar{x}_M^a\right)\right) \tag{12}$$

be the input truth vector, where $(x_i^a,\bar{x}_i^a)$ denotes the truth value of the i-th input node. Let

$$\mathbf{y} = \left(\left(y_1,\bar{y}_1\right),\left(y_2,\bar{y}_2\right),\ldots,\left(y_S,\bar{y}_S\right)\right) \tag{13}$$

be the cluster truth vector, where $(y_j,\bar{y}_j)$ denotes the truth value of the j-th cluster node. The size of **y** increases as the system learns novel mappings. Initially, $S = 0$, i.e., there is no cluster node. Let

$$\mathbf{x}^b = \left(\left(x_1^b,\bar{x}_1^b\right),\left(x_2^b,\bar{x}_2^b\right),\ldots,\left(x_N^b,\bar{x}_N^b\right)\right) \tag{14}$$

be the output truth vector, where $(x_k^b,\bar{x}_k^b)$ denotes the truth value of the k-th output node. Upon input presentation, $\mathbf{x}^a = \mathbf{A}$. During training, $\mathbf{x}^b = \mathbf{B}$. During testing, $\mathbf{x}^b$ is computed by the system.

Weight vectors: Each cluster node j is associated with a pair of adaptive input weight template $\mathbf{w}_j^a$ and output weight template $\mathbf{w}_j^b$. For a newly created node j, input weights $(w_{ji}^a,\bar{w}_{ji}^a) = (1/2M, -1/2M)$ and output weights $(w_{jk}^b,\bar{w}_{jk}^b) = (0,1)$.

Parameters: 3-valued NLN dynamics are determined by a choice parameter $\alpha > 0$; learning rates $\beta_a \in [0,1]$ and $\beta_b \in [0,1]$; and vigilance parameters $\rho_a \in [0,1]$ and $\rho_b \in [0,1]$.

Cluster selection: Given the input truth vector $\mathbf{x}^a$, for each cluster node j, the truth value $(y_j, \bar{y}_j)$ is computed by

$$(y_j, \bar{y}_j) = \mathscr{F}^c(t_j, \bar{t}_j), \tag{15}$$

where the positive truth t_j is given by

$$t_j = \sum_i w_{ji}^a x_i^a, \tag{16}$$

the negative truth $\bar{t}_j$ is given by

$$\bar{t}_j = \sum_i \bar{w}_{ji}^a \bar{x}_i^a, \tag{17}$$

and the choice threshold function $\mathscr{F}^c(\cdot,\cdot)$ is determined by

$$\mathscr{F}^c(t_j, \bar{t}_j) = \begin{cases} (1,0) & \text{if } j = J \text{ where } T_J > T_k \text{ for } k \neq J \\ (0,0) & \text{otherwise,} \end{cases} \tag{18}$$

where the net truth T_j is computed by $T_j = t_j - \bar{t}_j$.

Prediction match or reset: Prediction is confirmed if the *match degrees*, m_J^a and m_J^b, meet the vigilance criteria in the input and output layers respectively:

$$m_J^a = \frac{|\mathbf{x}^a \wedge {}^a\mathbf{w}_J^a|}{|\mathbf{x}^a|} \geq \rho_a \text{ and } m_J^b = \frac{|\mathbf{x}^b \wedge {}^b\mathbf{w}_J^b|}{|\mathbf{x}^b|} \geq \rho_b, \tag{19}$$

where the input match operator $\wedge^a$ is defined by

$$(\mathbf{x}^a \wedge {}^a\mathbf{w}_J^a)_i = \begin{cases} (1,0) & \text{if } x_i^a w_{Ji}^a > 0 \\ (0,-1) & \text{if } \bar{x}_i^a \bar{w}_{Ji}^a < 0 \\ (0,0) & \text{otherwise,} \end{cases} \tag{20}$$

the output match operator $\wedge^b$ is defined by

$$(\mathbf{x}^b \wedge {}^b\mathbf{w}_J^b)_k = \begin{cases} (1,0) & \text{if } w_{Jk}^b(x_k^b - \bar{x}_k^b) + \bar{w}_{Jk}^b > 0 \\ (0,0) & \text{otherwise,} \end{cases} \tag{21}$$

and the norm function $|\cdot|$ is defined by

$$|\mathbf{p}| = \sum_i (p_i - \bar{p}_i). \tag{22}$$

The input match operator ($\wedge^a$) checks if an input attribute value has the same sign as its corresponding weight value. The total number of input attributes that match (in sign) with their corresponding weight values is then normalized by the number of non-zero input attributes to yield the input match degree (m_J^a). The output match degree (m_J^b) is computed in a similar manner based on the output match operator ($\wedge^b$).

When both vigilance criteria are satisfied, learning ensues, as defined below. If any of the vigilance constraints is violated, mismatch reset occurs in which node J is shut down for the duration of the input presentation. The search process repeats to select another a new index J until a prediction match is achieved. If no such node exists, a new node J is created.

Learning: Once the search ends, the input weight vector $\mathbf{w}_J^a$ is updated according to the equation

$$\left(w_{Ji}^a, \overline{w}_{Ji}^a\right)^{(\text{new})} = (1-\beta_a)\left(w_{Ji}^a, \overline{w}_{Ji}\right)^{(\text{old})} + \beta_a \frac{\left(\mathbf{x}^a \wedge {}^a\mathbf{w}_J^{a(\text{old})}\right)_i}{\alpha + |\mathbf{x}^a \wedge {}^a\mathbf{w}_J^{a(\text{old})}|} \tag{23}$$

and the output weight vector $\mathbf{w}_J^b$ is updated according to the equation

$$\left(w_{Jk}^b, \overline{w}_{Jk}^b\right)^{(\text{new})} = (1-\beta_b)\left(w_{Jk}^b, \overline{w}_{Jk}^b\right)^{(\text{old})} + \beta_b\left(x_k^b - \bar{x}_k^b\right)\left(\mathbf{x}^b \wedge {}^b\mathbf{w}_J^{b(\text{old})}\right)_k. \tag{24}$$

For an input attribute i that has a positive or negative truth value in the truth value vector $\mathbf{x}^a$, the input weight learning equation increases the corresponding weight value in $(w_{Ji}^a, \overline{w}_{Ji}^a)$. For all other weight values, the learning equation reduces them towards zeroes. The output weight learning modifies only the positive weight values. The equation increases w_{Jk}^b towards 1 if the output attribute k has a positive truth value, decreases w_{Jk}^b towards -1 if the output attribute k has a negative truth value, and towards zero otherwise. For efficient coding of noisy input sets, it is useful to set $\beta_a = \beta_b = 1$ when J is a newly created node, and then take $\beta_a < 1$ and $\beta_b < 1$ after that. Fast *learning* corresponds to setting $\beta_a = \beta_b = 1$ for all existing cluster nodes.

Output prediction: During testing, given the cluster truth vector $\mathbf{y}$, the output truth vector $\mathbf{x}^b$ is determined by

$$\left(x_k^b, \bar{x}_k^b\right) = \mathscr{T}\left(\sum_j w_{jk}^b y_j, \sum_j \overline{w}_{jk}^b \bar{y}_j\right), \tag{25}$$

where the double step threshold function $\mathscr{T}$ is as defined in Eq. (4).

Match tracking: Match tracking rule as used in the ARTMAP search and prediction process [3] is useful in minimizing the number of nodes in the cluster layer. At the start of each input presentation, the vigilance parameter ρ_a equals a baseline vigilance ρ_a. If a reset due to a mismatch in the output layer occurs in the cluster layer, ρ_a is increased until it is slightly larger than the match degree m_J^a. The search process then selects another cluster node J under the revised vigilance criterion.

3.2. Fuzzy NLN

Fuzzy NLN extends the ordered-pairs of 0 and 1 in 3-valued NLN to ordered-pairs of real numbers between 0 and 1. Formally, the truth value domain $\mathscr{A}$ of fuzzy NLN is defined by

$$\mathscr{A} = \{(x, \bar{x}) : x \in [0,1], \bar{x} \in [0,1], x + \bar{x} = 1\}. \tag{26}$$

The SCM algorithm for fuzzy NLN follows closely to that of 3-valued NLN, but introduces a vector normalization step for cluster selection and match computation. The detailed algorithm is presented below.

Input and output patterns: Let $(a_i, \bar{a}_i)$ denote the i-th attribute value of the input vector **A** and let $(b_k, \bar{b}_k)$ denote the k-th attribute value of the output vector **B**.

Truth value vectors: Let

$$\mathbf{x}^a = \left(\left(x_1^a, \bar{x}_1^a\right), \left(x_2^a, \bar{x}_2^a\right), \ldots, \left(x_M^a, \bar{x}_M^a\right)\right) \tag{27}$$

be the input truth vector, where $(x_i^a, \bar{x}_i^a)$ denotes the truth value of the i-th input node. Let

$$\mathbf{y} = \left(\left(y_1, \bar{y}_1\right), \left(y_2, \bar{y}_2\right), \ldots, \left(y_S, \bar{y}_S\right)\right) \tag{28}$$

be the cluster truth vector, where $(y_j, \bar{y}_j)$ denotes the truth value of the j-th cluster node. The size of **y** increases as the system learns novel mappings. Initially, $S = 0$. Let

$$\mathbf{x}^b = \left(\left(x_1^b, \bar{x}_1^b\right), \left(x_2^b, \bar{x}_2^b\right), \ldots, \left(x_N^b, \bar{x}_N^b\right)\right) \tag{29}$$

be the output truth vector, where $(x_k^b, \bar{x}_k^b)$ denotes the truth value of the k-th output node. Upon input presentation, $\mathbf{x}^a = \mathbf{A}$. During training, $\mathbf{x}^b = \mathbf{B}$. During testing, $\mathbf{x}^b$ is computed by the system.

Weight vectors: Each cluster node j is associated with a pair of adaptive input weight template $\mathbf{w}_j^a$ and output weight template $\mathbf{w}_j^b$.

Parameters: Fuzzy NLN dynamics are determined by learning rates $\beta_a \in [0,1]$ and $\beta_b \in [0,1]$; and vigilance parameters $\rho_a \in [0,1]$ and $\rho_b \in [0,1]$.

Cluster selection: Given the input truth vector $\mathbf{x}^a$, for each cluster node j, the truth value $(y_j, \bar{y}_j)$ is computed by

$$\left(y_j, \bar{y}_j\right) = \mathscr{F}^c\left(t_j, \bar{t}_j\right). \tag{30}$$

where the positive truth t_j is given by

$$t_j = \frac{\sum_i w_{ji}^a x_i^a}{|\mathbf{w}_j^a|_2 |\mathbf{x}^a|_2}, \tag{31}$$

the negative truth $\bar{t}_j$ is given by

$$\bar{t}_j = \frac{\sum_i \bar{w}_{ji}^a \bar{x}_i^a}{|\mathbf{w}_j^a|_2 |\mathbf{x}^a|_2}, \tag{32}$$

and the choice threshold function $\mathscr{F}^c(\cdot,\cdot)$ is determined by

$$\mathscr{F}^c\left(t_j, \bar{t}_j\right) = \begin{cases} (1,0) & \text{if } j = J \text{ where } T_J > T_k \text{ for } k \neq J \\ (0,0) & \text{otherwise,} \end{cases} \tag{33}$$

where the net truth T_j is computed by

$$T_j = t_j - \bar{t}_j. \tag{34}$$

Prediction match or reset: Prediction is confirmed if the *match degrees*, m_J^a and m_J^b, meet the vigilance criteria in the input and output layers respectively:

$$m_J^a = \frac{\sum_i \left(w_{Ji}^a x_i^a - \overline{w}_{Ji}^a \overline{x}_i^a \right)}{|\mathbf{w}_J^a|_2 |\mathbf{x}^a|_2} \geq \rho_a \text{ and } m_J^b = \frac{\sum_i \left(w_{Ji}^b x_i^b - \overline{w}_{Ji}^b \overline{x}_i^b \right)}{|\mathbf{w}_J^b|_2 |\mathbf{x}^b|_2} \geq \rho_b, \tag{35}$$

where the L2-norm function $|\cdot|_2$ is defined by

$$|\mathbf{p}|_2 = \sqrt{\sum_i \left(p_i^2 + \overline{p}_i^2 \right)}. \tag{36}$$

Learning then ensues, as defined below. If any of the vigilance constraints is violated, mismatch reset occurs in which node J is shut down for the duration of the input presentation. The search process repeats to select another new index J until a prediction match is achieved. If no such node exists, a new node J is created.

Learning: Once the search ends, the input weight vector $\mathbf{w}_J^a$ is updated according to the equation

$$\left(w_{Ji}^a, \overline{w}_{Ji}^a \right)^{(\text{new})} = (1 - \beta_a) \left(w_{Ji}^a, \overline{w}_{Ji}^a \right)^{(\text{old})} + \beta_a \left(x_i^a, -\overline{x}_i^a \right) \tag{37}$$

and the output weight vector $\mathbf{w}_J^b$ is updated according to the equation

$$\left(w_{Jk}^b, \overline{w}_{Jk}^b \right)^{(\text{new})} = (1 - \beta_b) \left(w_{Jk}^b, \overline{w}_{Jk}^b \right)^{(\text{old})} + \beta_b \left(x_k^b, -\overline{x}_k^b \right). \tag{38}$$

Roughly speaking, the two learning equations adjust the weight vectors $\mathbf{w}_J^a$ and $\mathbf{w}_J^b$ towards the truth value vectors $\mathbf{x}_J^a$ and $-\mathbf{x}_J^b$ respectively. For efficient coding of noisy input sets, it is useful to set $\beta_a = \beta_b = 1$ when J is a newly created node, and then take $\beta_a < 1$ and $\beta_b < 1$ after that. The fast learning option is not advisable for fuzzy NLN.

Output prediction: During testing, given the cluster truth vector $\mathbf{y}$, the output truth vector $\mathbf{x}^b$ is determined by

$$\left(x_k^b, \overline{x}_k^b \right) = \mathscr{F}^s \left(\sum_j w_{jk}^b y_j, \sum_j \overline{w}_{jk}^b \overline{y}_j \right), \tag{39}$$

where the slope threshold function $\mathscr{F}^s$ is defined by

$$\mathscr{F}^s(t, \bar{t}) = \begin{cases} (1,0) & \text{if } t - \bar{t} \geq 1 \\ (0,1) & \text{if } t - \bar{t} \leq -1 \\ (t, 1-t) & \text{otherwise.} \end{cases} \tag{40}$$

Distributed prediction: In NLN systems using the choice threshold function (33) in the cluster layer, only the cluster node J that receives maximal input T_J predicts output. Another strategy is to use the choice rule during the initial period of supervised learning. However, during performance, a less extreme contrast enhanced cluster vector $\mathbf{y}$ is used in computing the output vector $\mathbf{x}^b$. Two algorithms that approximate contrast enhancement by competitive networks [7] are studied below.

Power rule: The power rule, as used in the ART-EMAP system [4,5], raises the input T_j of the j-th cluster node to a power p and normalizes the total activity:

$$(y_j, \bar{y}_j) = \mathscr{T}^p(t_j, \bar{t}_j) = \left(\frac{(T_j)^p}{\Sigma_k (T_k)^p}, 0 \right). \tag{41}$$

The power rule converges toward the choice rule as p becomes large.

K-max rule: In the spirit of the K nearest neighbor (KNN) system, the K-max rule picks the set of K out of the C cluster nodes with the largest input T_j for prediction. The cluster layer truth values $(y_j, \bar{y}_j)$ are then:

$$(y_j, \bar{y}_j) = \mathscr{T}^K(t_j, \bar{t}_j) = \begin{cases} \left(\dfrac{T_j}{\Sigma_{k \in \Phi} T_k}, 0 \right) & \text{if } j \in \Phi \\ (0,0) & \text{otherwise,} \end{cases} \tag{42}$$

where Φ is the set of K clusters with the largest T_j values. The K-max rule with $K = C$ is equivalent to the power rule with $p = 1$.

4. Comparative experiments

4.1. Mushroom classification

The mushroom classification problem is to determine whether a mushroom is edible or poisonous based on its observable features. The mushroom database [13] consists of 8124 instances, each of which is characterized by 22 nominal features. There are 3916 poisonous mushrooms, constituting 48.2% of the total population.

On this problem, Fu [25] used 1000 inputs to train a back-propagation network containing 127 input nodes, 63 hidden nodes, 2 output nodes, and 8127 connections. The network classified the 1000 training cases with 100% accuracy and a disjointed test set of 1000 cases with 99.0% accuracy.

In NLN simulations, the 22 nominal features are converted into 126 binary attributes. Both Boolean NLN and 3-valued NLN are trained using the SCM algorithm with the following parameter values: $\alpha = 0.001$, $\beta_a = \beta_b = 1$, $\bar{\rho}_a = 0$, and $\rho_b = 1$. The simulation results averaged over 20 runs are summarized in Table 1.

Table 1
Boolean and 3-valued NLN performance on the mushroom data compared with that of back-propagation network

Model	Train/test	No. epochs	No. nodes	Test accuracy (%)
Back-propagation	1000/1000	$O(100)$	63	99.0
Boolean NLN	1000/7124	3.2	16.6	99.7
3-valued NLN	1000/7124	2.3	4.9	99.7
3-valued NLN	2000/6124	2.4	5.6	99.9
3-valued NLN	3000/5124	2.4	5.7	100.0

After training on 1000 examples, Boolean NLN creates an average of 16.6 nodes and obtains 99.7% test accuracy on the remaining cases. When 3-valued NLN is trained with 1000 cases, an average of only 5.8 nodes are created, compared with the 63 hidden nodes of back-propagation network and 16.6 nodes of Boolean NLN. An accuracy of 99.7%, equivalent to that of Boolean NLN, is also obtained on the remaining 7124 cases. When a 3-valued NLN is given 2000 training cases, it creates roughly one more cluster node and pushes the test accuracy to 99.9%. Given 3000 or more cases, the number of clusters stabilizes at around 5.7 and the test accuracy converges at 100%.

We also trained 3-valued NLN using the construction algorithm (Section 2). The network with 1000 hidden nodes classifies the 1000 training pattern perfectly, but, due to the strict firing condition, is not able to generalize to a single test pattern. Even after applying feature enhancement and fine tuning [11,14], the best test accuracy is merely 47.3%, much lower than that obtained by back-propagation network or by SCM. The experiment confirms that although the construction method serves to prove the existence of an NLN to learn any arbitrary mapping, it does not produce a network that generalizes well to novel patterns. The SCM algorithm, on the other hand, produces a much superior generalization performance with a small number of hidden nodes.

4.2. Sonar signal recognition

The sonar return data set [6] contains 208 instances with 60 real-valued features, of which 97 instances are returns from roughly cylindrical rocks and 111 instances are returns from metal cylinders. This is a relatively difficult domain as the number of training examples is small and the data contain noises. In Gorman and Sejnowski's aspect angle dependent experiments, the data set was divided into a 104-element training set and a 104-element test set, with balanced representation in each aspect angle. After learning the training set, perceptron classifies only 73% of the test set patterns correctly (Table 2). Back-propagation network with 12 hidden nodes obtains a test set accuracy of 90.4%. Increasing the number of hidden nodes to 24, however, degrades the performance.

The sonar return data set has also been used to evaluate other learning systems, including K nearest neighbor (KNN), fuzzy ARTMAP, and fuzzy ARAM [15]. The KNN system that stores all training patterns, performs best with $K = 1$, producing a test

Table 2
Fuzzy NLN performance on the sonar return data comparing with alternative learning systems

Model	No. epochs	No. nodes	Test accuracy (%)
Perceptron	300	0	73.1
Back-propagation	300	12	90.4
Back-propagation	300	24	89.2
KNN	1	104	91.6
Fuzzy ARTMAP	8–34	22–42	91.6
Fuzzy ARAM	2	68–72	92.9
Fuzzy NLN	4.3	55.3	92.4
Fuzzy NLN (power = 200)	4.3	55.3	93.2

set performance of 91.6%. This is slightly better than the test accuracy of back-propagation network. Fuzzy ARAM with ARTMAP configuration produces the same level of accuracy as KNN with only 22 to 42 category nodes. The number of learning iterations ranges from 8 to 34, about ten times less than that of back-propagation networks. Fuzzy ARAM with fast learning and high vigilance converges in merely two iterations. Also, a better prediction rate is obtained at 92.9%.

Fuzzy NLN experiments are conducted with the following parameter values: $\beta_a = 0.15$, $\beta_b = 1.0$, $\rho_a = 0.9$, and $\rho_b = 1.0$. The fuzzy NLN performance, averaged over 10 runs, is also summarized in Table 2. Fuzzy NLN takes an average of 4.3 iterations to learn the training set. The test accuracy of fuzzy NLN is slightly lower than that of fuzzy ARAM, but so is the number of hidden nodes (clusters). Experiments are also conducted using the power rule with $p = 50,100,150,\ldots$ in a step size of 50. The test accuracy peaks at 93.2% with $p = 200$ and converges towards the performance of choice (92.4%) as p becomes larger.

4.3. Sunspots prediction

The sunspots series consists of yearly averages of sunspots starting from the year 1700 to the year 1979. The sunspot prediction problem is a typical time series prediction problem, in which we want to predict the sunspots number for the following year, based on a *forecast window* consisting of consecutive sunspots data in the past few years.

In the following sections, we shall first define the performance measures that we used to evaluate a time series predictor and the encoding schemes used in our simulations. We then report our simulation results and compare them with two existing time series predictors.

4.3.1. Performance measures

Given a time series data, we extract *examples*, each consisting of a window of m consecutive values followed by a *target* value. The last value in the forecast window, preceding the target value, is also called the *current* value. The aim of a time series predictor is to make its *predicted* value as close as possible to the target value, based on the forecast window.

Let c_j, t_j, and p_j denote the current, target, and predicted values of an example e_j respectively. Also, let c_m, t_m, and p_m denote the mean of the current, target, and predicted values averaged over all examples, respectively. We compute the following indices to evaluate the performance of a time series predictor.

Mean squared error, given by

$$\mathrm{MSE} = \frac{\Sigma_j (t_j - p_j)^2}{S} \geq 0, \tag{43}$$

is the classical cost function for regression systems, divided by the number of patterns in the data set (S), so as to make it independent of the size of the data set.

Normalized mean squared error, given by

$$\mathrm{NMSE} = \frac{\Sigma_j (t_j - p_j)^2}{\Sigma_j (t_j - t_m)^2} \geq 0, \tag{44}$$

indicates the closeness of the prediction to the target. By virtue of the normalization, it is independent of both the size and the dynamic range of the data set.

Correlation coefficient, given by

$$\mathrm{CC} = \frac{\Sigma_j (t_j - t_m)(p_j - p_m)}{\sqrt{\Sigma_j (t_j - t_m)^2 \Sigma_j (p_j - p_m)^2}} \in [-1,1], \tag{45}$$

measures how well the predicted values correlates with the target values. It roughly corresponds to the converse of the Normalized Mean Squared Error.

Lag coefficient, given by

$$\mathrm{LC} = \frac{\Sigma_j (c_j - c_m)(p_j - p_m)}{\sqrt{\Sigma_j (c_j - c_m)^2 \Sigma_j (p_j - p_m)^2}} \in [-1,1], \tag{46}$$

measures the extent to which the prediction lags behind the target value by computing its correlation with the current value.

Information coefficient, given by

$$\mathrm{IC} = \sqrt{\frac{\Sigma_j (p_j - t_j)^2}{\Sigma (c_j - t_j)^2}} \geq 0, \tag{47}$$

measures the performance with respect to a trivial predictor that uses the current value as the prediction. An information coefficient of 1 means that the performance is only as good as that of the trivial predictor; a value of less than 1, on the other hand, means that it is better.

Trend prediction accuracy (Trend) gives the percentage of rise and fall of the predicted values that match with those of the target values. In other words, it measures how well the prediction follows the trend of the actual series.

4.3.2. Encoding schemes

Given an example e_j consisting of a forecast window $\{v_1, v_2, \ldots, v_m\}$ and a target value v_{m+1}, three possible input configurations are listed below.

Absolute encoding (A) uses the actual values in the forecast window. The input vector **a** is given by

$$\mathbf{a} = (v_1, v_2, \ldots, v_m). \tag{48}$$

Difference encoding (D) uses the consecutive differences between adjacent values. The input vector **a** is given by

$$\mathbf{a} = (v_2 - v_1, v_3 - v_2, \ldots, v_m - v_{m-1}). \tag{49}$$

Reference encoding (R) takes the difference between each forecast window value and the current value (v_m). The input vector **a** is given by

$$\mathbf{a} = (v_m - v_1, v_m - v_2, \ldots, v_m - v_{m-1}). \tag{50}$$

For the case of predicting a single output value, reference encoding is equivalent to difference encoding. The two possible output configurations are listed below.

Absolute encoding (A) uses the actual target value. The output vector **b** is given by

$$\mathbf{b} = (v_{m+1}). \tag{51}$$

Difference encoding (D) uses the difference between the target value and the current value. The output vector **b** is given by

$$\mathbf{b} = (v_{m+1} - v_m). \tag{52}$$

Hence, there are a total of six possible encoding schemes, each of which is a different combination of input-output configurations. All input and output values are normalized to the interval [0,1].

4.3.3. Simulations and results

In the previous experiments [23,24], yearly sunspots data from 1700 through 1920 were used for training and the data from 1921 through 1979 were used for evaluation of the prediction. In addition, the prediction set was split into two parts, 1921–1955 (for validation) and 1956–1979 (for testing). In order to compare with previous results, we fix the forecast window size at 12 and use the same training set. Since SCM does not need validation, we use the entire prediction set for testing.

Fuzzy NLN simulations are conducted using the following parameter values: $\rho_a = 0.0$, $\rho_b = 0.9$, and $\epsilon = 0.01$. We conduct experiments using the K-max rule in which K is varied from 1 to 5, and find that the best performance is obtained when $K = 2$. For each encoding scheme, different learning rates β_a and β_b from $\{0.1, 0.2, \ldots, 0.5\}$ are tried and the best performance is picked from these trials. The simulation results are summarized in Table 3, where the best entry for each performance measure (column) is highlighted with boldface.

Fuzzy NLN gives the best overall performance when the actual forecast window values are used to approximate the difference between the current and forecast values (i.e., configuration AD). In general, using the difference between the current and forecast values as the network output to compute the forecast prediction (i.e., output configuration D) gives better performance than using the actual forecast value. Fig. 4 shows the fuzzy NLN predictions against the actual sunspots series.

Table 3
Performance of fuzzy NLN, in terms of six performance measures, using various input-output configurations

Encoding scheme		Performance measures						
Input	Output	No. nodes	MSE	NMSE	CC	LC	IC	Trend (%)
A	A	38	0.022	0.334	0.860	0.442	0.912	89.8
D	A	**24**	0.023	0.349	0.873	**0.366**	0.931	86.4
R	A	26	0.028	0.425	0.820	0.419	1.028	89.8
A	D	39	**0.011**	**0.165**	**0.914**	0.526	**0.640**	**96.6**
D	D	44	0.013	0.199	0.897	0.552	0.703	93.2
R	D	28	0.014	0.210	0.892	0.564	0.721	91.5

Table 4
Performance comparison of fuzzy NLN with the back-propagation network and the threshold autoregressive model, in terms of normalized mean square error, on the sunspot data

Model	Training (1700–1920)	Validation (1921–1955)	Testing (1956–1979)
Back-propagation	0.082	0.086	0.35
TAR	0.097	0.097	0.28
Fuzzy NLN (AD)	0.179	0.140	0.21

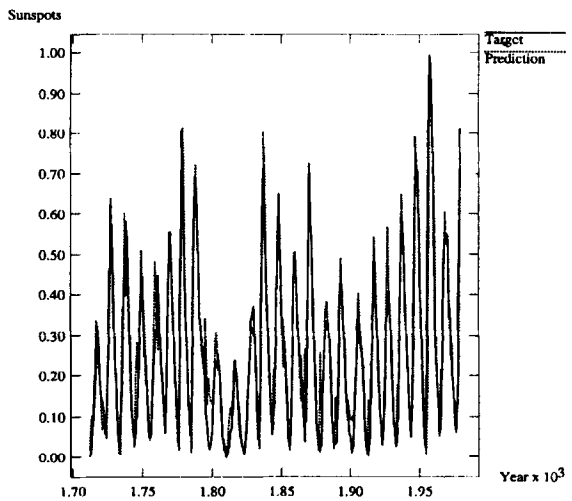


Fig. 4. Fuzzy NLN sunspot prediction against the actual series. Data from 1700 to 1920 are used for training and data from 1921 to 1979 are used for testing.

Experiments on the sunspots data have been conducted using the back-propagation network [24] and the threshold autoregressive (TAR) model [23], which both refer to normalized mean squared error as *average relative variance*. Table 4 compares fuzzy NLN with these two models based on this measure. For proper comparison, we only look at the results for the test set (1956–1979), since the performance for the validation set (1921–1955) has been optimized during training in both the back-propagation network and the threshold autoregressive (TAR) model.

5. Conclusions and extensions

We have presented a Supervised Clustering and Matching (SCM) algorithm for Neural Logic Network to perform inductive learning and pattern matching. By adopting a match-based clustering approach to perform supervised learning, several desired features are achieved. The most notable advantage over gradient descent algorithms is *speed*. Using the SCM algorithm, the network weights typically converge in a few

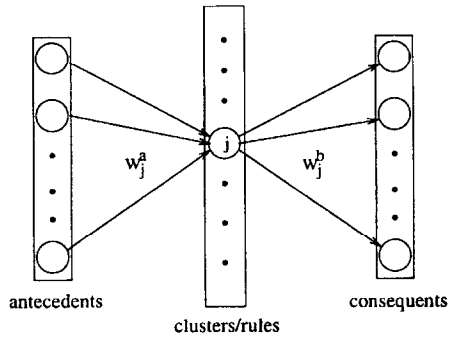


Fig. 5. Rules in an inductive NLN. Each cluster node j corresponds to a rule that maps a prototype feature vector $\mathbf{w}_j^a$ (antecedents) to a prediction vector $\mathbf{w}_j^b$ (consequents).

iterations. Although the system allows incremental learning, the network size, in terms of the number of hidden (cluster) nodes, is manageable. In certain problems, such as mushroom classification, the cluster based learning approach even leads to more compressed internal representation. In terms of predictive performance, the SCM algorithm also compares favorably with alternative learning systems, including K nearest neighbor, fuzzy ARTMAP, fuzzy ARAM, back-propagation network, and threshold autoregressive model.

While it is difficult to explain for specific applications why SCM performs better than the gradient descent error back-propagation (BP) algorithms, it is important to note that SCM adopts an approach fundamentally different from BP to learning pattern mappings. Whereas BP classifies patterns by separating them using hyperplanes, SCM classifies patterns by grouping them into clusters. Whereas BP learns when an error arises, SCM learns when a match occurs. Depending on the pattern distribution and the number of the training patterns available, one algorithm could generalize better than the other one.

Another important advantage of the inductive NLN architecture is the ease of interpreting the learned knowledge. In an inductive NLN network, each node in the cluster layer encodes a pair of input and output template patterns. Learned weight vectors, one for each cluster node, thus correspond to a set of rules that link antecedents to consequents (Fig. 5). This type of knowledge structure allows the translation of the network architecture to a set of symbolic rules that can be interpreted easily [5]. In addition, symbolic rules can be inserted into an inductive NLN before learning and refined using the SCM learning algorithm. This gives rise to an integrated system that processes both inductive and deductive knowledge.

Acknowledgements

The authors gratefully acknowledge all members of the RWCP Neuro ISS laboratory, especially Hoon-Heng Teh, Ho-Chung Lui, Liya Ding, and Joo-Hwee Lim for many stimulating ideas and discussions. The authors also thank the two anonymous reviewers for many useful comments and suggestions to a previous version of this manuscript.

References

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

[2] G.A. Carpenter and S. Grossberg, ART 2: Self-organization of stable category recognition codes for analog input patterns, *Applied Optics* 26 (1987) 4919–4930.

[3] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen, Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, *IEEE Transactions on Neural Networks* 3 (1992) 698–713.

[4] G.A. Carpenter and W.D. Ross, ART-EMAP: A neural network architecture for object recognition by evidence accumulation network, *Proc. World Congress on Neural Networks, Portland, OR* (1993) 649–656.

[5] G.A. Carpenter and A.H. Tan, Rule extraction: From neural architecture to symbolic representation, *Connection Science* 7 (1995) 3–27.

[6] R.P. Gorman and T.J. Sejnowski, Analysis of hidden units in a layered network trained to classify sonar targets, *Neural Networks* 1 (1988) 75–89.

[7] S. Grossberg, Contour enhancement, short term memory, and constancies in reverberating neural networks, *Studies in Applied Mathematics* 52 (1973) 217–257.

[8] T. Kohonen, *Self-organization and Associative Memory* (Springer-Verlag, 1988).

[9] J.H. Lim, Incremental case-based pattern classifier, *Proc. International Conference on Artificial Neural Networks* (Amsterdam, 1993).

[10] J.H. Lim and H.H. Teh, Complex neural logic networks and handwriting character recognition, Technical Report, Real World Computing Partnership, 1995.

[11] B.T. Low, H.C. Lui, A.H. Tan, and H.H. Teh, Connectionist expert system with adaptive learning capability, *IEEE Transactions on Knowledge and Data Engineering* 3(2) (1991) 200–207.

[12] P.M. Murphy and D.W. Aha, UCI repository of machine learning databases [machine-readable data repository, University of California, Department of Information and Computer Science, Irvine, CA, 1992.

[13] J.S. Schlimmer, Concept acquisition through representational adjustment, Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1987.

[14] A.H. Tan, A neural logic environment for expert systems, Master Thesis, Department of Information Systems and Computer Science, National University of Singapore, 1991.

[15] A.H. Tan, Adaptive resonance associative map, *Neural Networks* 8(3) (1995) 437–446.

[16] A.H. Tan and H.H. Teh, Connectionist expert systems: An inductive cum deductive approach, *Information Technology* 3(1) (1989) 63–72.

[17] A.H. Tan and L.N. Teow, Neural logic networks for pattern recognition, time series prediction, and knowledge integration, *Proc. Real World Computing '95 Joint Symposium, Tokyo* (1995) 67–68.

[18] A.H. Tan and L.N. Teow, Learning by supervised clustering and matching, *Proc. 1995 IEEE International Conference on Neural Networks, Vol 1* (Perth, 1995) 242–246.

[19] H.H. Teh, A new class of neural networks called neural logic networks: First technical report, Technical Report TR-93004, Real World Computing Partnership, 1993.

[20] H.H. Teh, A new class of neural networks called neural logic networks: Second technical report, Technical Report TR-94013, Real World Computing Partnership, 1994.

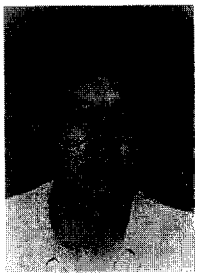
[21] H.H. Teh, *Neural Logic Networks* (World Scientific Publishing Company, 1995).

[22] H.H. Teh and A.H. Tan, Connectionist expert systems: A neural logic models' approach, *Proc. Inter-faculty Seminar on Neuronet Computing* (National University of Singapore, 1989) 16–32.

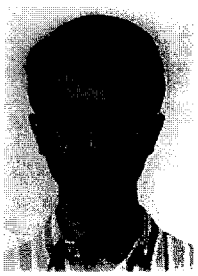
[23] H. Tong and K.S. Lim, Threshold autoregression, limit cycles and cyclical data, *J.R. Stat. Soc. B* 42 (1980) 245.

[24] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart, Predicting the future: A connectionist approach, *International Journal of Neural Systems* 1(3) (1990) 193–209.

[25] L.M. Fu, A neural network model for learning rule-based systems, *Proc. Int. Joint Conf. on Neural Networks*, Vol. 1 (Baltimore, 1992) 343–348.



Ah-Hwee Tan is an associate researcher at the Institute of Systems Science. He received his Ph.D. degree in Cognitive and Neural Systems from Boston University in 1994. Prior to that, he obtained his B.Sc. (Hons) in 1989 and M.Sc. in 1991, both in Computer and Information Science from the National University of Singapore. He has published over 25 technical papers related to neural networks and knowledge based systems. His current research interests include integration of neural network and symbolic knowledge processing, cognitive modeling, agent architecture, computational learning theory, pattern recognition, and associative memory. He is a member of the International Neural Network Society (INNS) and the Singapore Computer Society.



Loo-Nin Teow received his B.Sc. degree in Computer and Information Science in 1992 and his post-graduate diploma in Computing Technology in 1995, both from the National University of Singapore. He is presently a software engineer at the Institute of Systems Science. His research interests include neural networks, fuzzy logic, hybrid systems, and character recognition.