Supervised learning on a fuzzy Petri net

Amit Konar a, Uday K. Chakraborty b,*, Paul P. Wang c

a Electronics and Tele-Communication Engineering Department, Jadavpur University, Calcutta 700032, India
b Department of Mathematics and Computer Science, One University Boulevard, University of Missouri, St. Louis, MO 63121, USA
c Electrical and Computer Engineering Department, Duke University, Durham, NC 27708, USA

Received 1 July 2003; received in revised form 25 May 2004; accepted 26 May 2004

Abstract

Feed-forward neural networks used for pattern classification generally have one input layer, one output layer and several hidden layers. The hidden layers in these networks add extra non-linearity for realization of precise functional mapping between the input and the output layers, but semantic relations of the hidden layers with their predecessor and successor layers cannot be justified. This paper presents a novel scheme for supervised learning on a fuzzy Petri net that provides semantic justification of the hidden layers, and is capable of approximate reasoning and learning from noisy training instances. An algorithm for training a feed-forward fuzzy Petri net and an analysis of its convergence have been presented in the paper. The paper also examines the scope of the learning algorithm in object recognition from 2D geometric views.

© 2004 Elsevier Inc. All rights reserved.
1. Introduction

Fuzzy Petri nets (FPNs) [1–7,9–13,18,19] have been used for knowledge representation and reasoning in the presence of inexact data and knowledge bases. Machine learning with fuzzy AND-OR neurons [8,14,16] and with fuzzy Petri nets [17] have been proposed by Pedrycz. In [15], while modeling a specialized cognitive structure, Pedrycz examined the scope of the model in fuzzy pattern recognition. The present paper addresses an alternative scheme of supervised learning on fuzzy Petri nets and studies its application in pattern recognition. Our method combines the power of high level reasoning of the FPN and the ease of training of the supervised feed-forward neural net.

The proposed model of fuzzy Petri net comprises fuzzy OR and AND neurons represented by places and transitions of the network. Normally, a set of transitions followed by a set of places constitutes a layer. An \( l \)-layered fuzzy Petri net thus contains \( l-1 \) layers of transitions followed by places, and an additional input layer consisting of places only. The places in the last layer are called concluding places. Such a network has two representational benefits. First, it can represent inexact knowledge like conventional FPNs. Second, the network can be trained with a set of input–output patterns (as in a feed-forward neural net). Such a network, when used for object recognition from fuzzy features, offers the benefits of both inexact reasoning and machine learning on a common platform.

The paper starts with a formal description of the proposed model of the fuzzy Petri net. An algorithm for adaptation of transition thresholds from a given set of training instances is then presented, with applications in object recognition. An analysis of the proposed training algorithm shows its unconditional convergence.

This paper is organized in six sections. The proposed model of fuzzy Petri net is covered in Section 2. The algorithm for training the fuzzy Petri net with \( n \) sets of input–output training instances is presented in Section 3. Section 4 provides an analysis of convergence of the algorithm. An illustrative case study, highlighting the training with 2-D geometric objects and their recognition is presented in Section 5. Finally, conclusions are drawn in Section 6.

2. Proposed model of fuzzy Petri net

The principles of supervised learning to be introduced in this paper have been developed based on the following model of FPN.

Definition 1. An \( FPN \) is a 9-tuple, given by

\[
FPN = (P, Tr, T, D, I, O, Th, n, W)
\]
where \( P = (p_1, p_2, \ldots, p_n) \) is a finite set of places; \( T_r = (t_1, t_2, \ldots, t_m) \) is a finite set of transitions; \( T = (t_1, t_2, \ldots, t_m) \) is a set of fuzzy truth tokens in the interval \([0, 1]\) associated with the transitions \((t_1, t_2, \ldots, t_m)\), respectively; \( D = (d_1, d_2, \ldots, d_n) \) is a finite set of propositions, where proposition \( d_k \) corresponds to place \( p_k \); \( P \cap T_r \cap D = \emptyset \); cardinality of \((P) = \text{cardinality of } (D)\); \( I: T_r \to P^\infty \) is the input function, representing a mapping from transitions to bags of (their input) places; \( O: T_r \to P^\infty \) is the output function, representing a mapping from transitions to bags of (their output) places; \( T_h = (th_1, th_2, \ldots, th_m) \) represents a set of threshold values in the interval \([0, 1]\) associated with transitions \((t_1, t_2, \ldots, t_m)\), respectively; \( n: P \to [0, 1] \) is an association function hereafter called fuzzy belief, representing a mapping from places to real values between 0 and 1; \( n(p_i) = n_i \) (say); \( W = w_{ij} \) is the set of weights from the \( j \)th transition to the \( i \)th place, where \( i \) and \( j \) are integers. A few more definitions, which will be referred to frequently while presenting the model, are in order.

**Definition 2.** If \( p_i \in I(t_a) \) and \( p_i \in O(t_b) \) then \( t_a \) is immediately reachable from \( t_b \) [4]. Again, if \( t_a \) is immediately reachable from \( t_b \) and \( t_b \) is immediately reachable from \( t_c \), then \( t_a \) is reachable from \( t_c \).

The reachability property is the reflexive, transitive closure of the immediate reachability property [4]. We would use \( \text{IRS}(t_a) \) and \( \text{RS}(t_a) \) operators to denote the set of transitions immediately reachable and reachable from transition \( t_a \), respectively.

Moreover, if \( t_a \in [\text{IRS}(\text{IRS}(\text{IRS} \cdots k\text{-times}(t_a))))] \), denoted by \( \text{IRS}^k(t_b) \), then \( t_a \) is reachable from \( t_b \) with a degree of reachability \( k \). For reachability analysis, two connectivity matrices \((P \text{ and } Q, \text{ defined below})\) will be needed.

**Definition 3.** A place to transition connectivity (PTC) matrix \( Q \) is a binary matrix whose element \( q_{jk} = 1 \), if for place \( p_k \) and transition \( t_j \), \( p_k \in I(t_j) \); otherwise \( q_{jk} = 0 \). If the FPN has \( n \) places and \( m \) transitions, then the \( Q \) matrix is of dimension \((m \times n)\) [9].

**Definition 4.** A transition to place connectivity (TPC) matrix \( P \) is a binary matrix whose element \( p_{ij} = 1 \), if for place \( p_i \) and transition \( t_j \), \( p_i \in O(t_j) \); otherwise \( p_{ij} = 0 \). With \( n \) places and \( m \) transitions in the FPN, the \( P \) matrix is of dimension \((n \times m)\) [10].

**Definition 5.** If \( p_{ij} \), the \((i, j)\)th element of matrix \( P \), is 1, then place \( p_i \) is said to be reachable from transition \( t_j \). We call this reachability transition-to-place reachability to distinguish it from transition-to-transition reachability.
Since the max–min composition \((Q \circ P)\) represents a mapping from transitions to their immediately reachable transitions, the presence of a 1 in the matrix \(M_1 = (Q \circ P)\) at position \((j, i)\) indicates that \(tr_j \in \text{IRS}(tr_i)\). Analogously, a 1 at position \((j, i)\) in matrix \(M_r = (Q \circ P)^r\) for positive integer \(r\), represents \(tr_j \in \text{IRS}^r(tr_i)\), i.e., \(tr_j\) is reachable from \(tr_i\) with a degree of reachability \(r\). Further, a 1 at position \((i, j)\) in the matrix \(P \circ (Q \circ P)^r\) denotes that there is a place \(p_i\) reachable from transition \(tr_j\) with a degree of reachability \(r\) from transition-to-transition and a degree of reachability one from transition-to-place.

**Definition 6.** A transition \(tr_j\) is **enabled** if \(p_i\) possesses fuzzy beliefs for \(\forall p_i \in I(tr_j)\).

An enabled transition fires by generating a **fuzzy truth token (FTT)** at its output arc. The value of the FTT is given by

\[
t_{j}(t + 1) = \begin{cases} 
\bigwedge_{i} \{ n_i | p_i \in I(tr_j) \} - \text{th}_j & \text{if } \bigwedge_{i} \{ n_i | p_i \in I(tr_j) \} > \text{th}_j \\
0 & \text{otherwise}
\end{cases}
\]

(1)

Since FTT computation at a transition involves taking fuzzy AND (min) of the beliefs of its input places, a transition may be regarded as an AND neuron.

**Definition 7.** After the firing of \(tr_j\)'s the **fuzzy belief** of \(n_k\), at place \(p_k\), where \(p_k \in O(tr_j) \forall j\), is given by

\[
n_{k}(t + 1) = n_{k}(t) \lor \left[ \bigvee_{j} \{t_{j}(t + 1)\} \right]
\]

(2)

Since belief computation involves the fuzzy OR (max) operation, the places may be regarded as OR neurons.

**Definition 8.** A set of transitions \(\{tr_x\}\) and a set of places \(\{p_y\}\), where \(\forall p \in \{p_y\}\) and \(\forall tr \in \{tr_x\}, p \in O(tr)\), constitute a **layer** in the proposed FPN.

**Definition 9.** The **input layer** in an FPN is a special layer which consists of only input places, and there does not exist any \(tr_j\) such that \(p \in O(tr_j)\).
2.1. State-space formulation

Let $N_t$ be the fuzzy belief vector, whose $i$th component $n_i$ denotes the belief of proposition $d_i$ located in place $p_i$ at time $t$. Also assume $T_t$ to be the FTT vector, whose $j$th component $t_j$ is the FTT of transition $tr_j$ at time $t$. Then the FTT update Eq. (1) for $m$ transitions together can be described [10] in state-space form as follows:

$$T_{t+1} = (Q \circ N_t^c)^c - Th \quad \text{if} \quad (Q \circ N_t^c)^c > Th$$ \hspace{1cm} (3)

where $Th$ denotes the threshold vector of transitions, such that its $i$th component $th_i$ is the threshold of transition $tr_i$ and $c$ over a vector denotes its component-wise one's complement.

The belief updating expression (2) of the entire FPN can also be represented in state-space form as follows:

$$N_{t+1} = N_t \lor (P \circ T_{t+1})$$
$$= N_t \lor [P \circ \{(Q \circ N_t^c)^c - Th\}] \quad \text{if} (Q \circ N_t^c)^c > Th$$ \hspace{1cm} (4)

For an FPN with $l$ layers, one may recursively update expression $N_{t+1}$ $l$-times in (4) to find the belief vector $N_l$ in terms of the initial belief vector $N_0$ as follows:

$$N_l = N_0 \lor \left[ P \circ \left\{ \bigvee_{i=1}^{l} \{(Q \circ N_i^c)^c - Th\} \right\} \right]$$ \hspace{1cm} (5)

It may be noted that the components of the initial belief vector $N_0$ for the input layer are non-zero, while other components of $N_0$ are set to zero.

The vector $N_l$ yields the fuzzy belief at all places in the neural FPN at steady-state. The components of the steady state belief vector $N_l$ corresponding to the places in the last layer may now be considered for estimation of error vector $E_k$. If the estimation of the error vector after one forward pass in the neural FPN followed by the adjustment of thresholds of transitions in a pre-defined layer is termed a single threshold adjustment cycle (TAC), then the error vector after the $k$th TAC is given by

$$E_k = D - A \circ N_l(k)$$ \hspace{1cm} (6)

where $D$ is the target vector corresponding to the places at the last layer, $N_l(k)$ is the $N_l$ vector after the $k$th TAC, and $A$ is a binary mask matrix of the following form:
Training in the proposed neural FPN refers to layer-wise adaptation of thresholds of the transitions. Unlike the classical back-propagation algorithm, training in the neural FPN begins at the input layer and is continued layer-wise until the last layer is reached. The training may further involve several phases of adaptation of thresholds in the entire network.

Before presenting the training algorithms, we briefly outline the principle of training. Given an input–output training instance, the neural FPN first computes the output through a forward pass in the network and evaluates the outputs of the last layer AND-neurons. It then evaluates the error vector by taking the component-wise difference between the computed output vector and the prescribed target vector (output instance). Let \( th_j \) be the threshold of a transition \( tr_j \) in a given layer and let \( e_u \) be the scalar error at the concluding place \( u \) in the output layer, such that \( p_u \) has connectivity from \( tr_j \). By “connectivity” we mean the following: if a concluding place \( p_u \in O(tr_k) \) then \( tr_k \in RS(tr_j) \). Then the threshold \( th_j \) is adapted using

\[
A = \begin{pmatrix}
    p_1 & p_2 & \cdots & p_m \\
    p_{m+1} & 0 & \cdots & 0 \\
    p_{m+2} & 0 & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    p_{m+n} & 0 & \cdots & 0 \\
\end{pmatrix}
\]

\[
\Phi = \begin{pmatrix}
    \Phi \\
    \Phi \\
    \Phi \\
\end{pmatrix}
\]

(7)
\[ \text{th}_j \leftarrow \text{th}_j - \sqrt{\eta_u(e_u)}. \] (8)

After the thresholds in a given layer are adapted by the above principle, the network undergoes a forward pass for re-evaluation of the error vector at the output layer and adaptation of thresholds for the transitions in the subsequent layer. The subsequent layer in the present context usually is the next layer, but it can be the first layer as well if the threshold adaptation is currently performed in the last layer. The whole scheme of training a neural FPN is represented in vector–matrix form as follows.

Recall that a TAC constitutes a forward pass in the network for computing the error vector at the output layer followed by threshold adjustments of all the transitions in a given layer. Let

- \( k \) be the total number of TACs so far performed + 1 (the 1 is due to the current TAC), and
- \( r \) be the total number of training cycles, i.e., the number of times training has so far been given to the entire network.

Considering \( l \) to be the number of layers in the network, we define \( z = k - r \times l \), where \( z \) denotes the layer currently selected for threshold adaptation. The definition of \( z \) follows from the definitions of \( k \) and \( r \). For convenience, we can re-write the expression as \( k = r \times l + z \). Then for any positive integer \( k \), \( z = k \) modulo \( l \). For example, let \( k = 8 \) and \( l = 3 \), then \( z = 8 \) modulo \( 3 = 2 \). This means that if the current TAC is the 8th cycle, and the FPN contains 3 layers, then the second layer is currently selected for threshold adaptation. It is important to note that \( r \) can be defined as \( k \) \ div \( l \). In the present example, \( 8 \) \ div \( 3 = 2 \), i.e., two training cycles have already elapsed.

In order to transfer the error vector \( E_k \) of the \( k \)th TAC to the layer \((k - r \times l)\), we need to define a matrix \( M \) that denotes the connectivity from the transitions at layer \((k - r \times l)\) to the places at the output layer:

\[ M = P \circ (Q \circ P)^{(k-r \times l)}. \] (9)

Consequently, \( M^T \) denotes the connectivity from the places in the output layer to the transitions at layer \((k - r \times l)\). Now, to adapt the thresholds at the \((k - r \times l)\)th layer, we need to construct a rotational partitioned matrix \( W_{k-r \times l} \), where partition matrix \( I \) occupies the \((k - r \times l)\)th position out of \( l \) possible partitions, and the remaining \((l-1)\) positions contain null matrices \( \phi \):

\[ W_{(k-r \times l)} = \begin{bmatrix} [\phi] \\ [I] \\ [\phi] \\ \vdots_{m \times s} \end{bmatrix}. \]

For example, for a 3-layered net, if \( l = 3 \), \( k = 1 \), \( r = 0 \), then \( W_{(k-r \times l)} \) would be
Similarly, for $l = 3, k = 2, r = 0$, we have

$$W_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} [I] \\ [\Phi] \end{bmatrix};$$

for $l = 3, k = 3, r = 0$, we have

$$W_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} [I] \\ [\Phi] \end{bmatrix};$$

for $l = 3, k = 4, r = 1$, we have

$$W_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} [I] \\ [\Phi] \end{bmatrix};$$

and so on.

Adaptation of thresholds at layer $k - r \times l$ can now be done easily using the following expression (Th is the $(m \times 1)$ threshold vector):

$$\text{Th} = \text{Th} - W_{(k-r \times l)} \circ M^T \circ E_k$$

where

$$M = P \circ (Q \circ P)^{(k-r \times l)}$$

and

$$W_{(k-r \times l)} = \frac{1}{\sqrt{2}} \begin{bmatrix} [\Phi] \\ [I] \\ [\Phi] \end{bmatrix}_{m \times s}$$

The algorithm for training a multi-layered feed-forward FPN with a single input–output pattern is presented below:

**Procedure Train_with_single_I/O_pattern** $(N_o, P, Q, D, \text{Th}_k)$

begin

$k := 1; \ text{error} \_\text{sum} := 1000; */\text{large error} \_\text{sum}*/; r := 0$;

while (error_sum $>$ pre-assigned_limit) AND ($r < r_{\text{max}}$) do

begin

$N_i(k) := N_o \\lor \left[ P \circ \left\{ \bigcup_{i=1}^l \{(Q \circ N^c_i)^c - \text{Th}\} \right\} \right];$

$E_k := D - A \circ N_i(k);$  

error_sum := $\sum_{u} (E_u)^{1/3}; //E_u = \text{components of } E_k \text{ at concluding places } p_u//$

Th := Th $- W_{(k-r \times l)} \circ M^T \circ E_k;$

end
where $M := P \circ (Q \circ P)^{(k-r \times \ell)}$

and $W_{(k-r \times \ell)} = \begin{bmatrix} \Phi \\ 1 \\ \Phi \end{bmatrix} \otimes_{m \times s} $;

if $\text{mod}(k/l) = 0$ then $r := r + 1$;

$k := k + 1$;

end while;

end.

The while loop in the above procedure computes the output vector $A \circ N_k(k)$ and determines the error vector $E_k$ as the difference of $A \circ N_k(k)$ from the target vector $D$. The $E_k$ is then used to layer-wise adapt the thresholds, starting from the first layer. Note that although the threshold vector $Th$ includes the thresholds of all the transitions present in the network, the adaptation rule changes the thresholds at the $(k-r \times \ell)$th layer only. A check of $\text{mod}(k/l) = 0$ is included at the end to test whether thresholds of all layers in the FPN have been updated. If yes, $r$ is increased by 1, indicating a fresh start of threshold adaptation beginning with the first layer. The TAC count $k$ is increased after each threshold adaptation in the layers of the FPN.

The performance of learning in the above procedure depends greatly on an important attribute called error-sum, which is computed by summing the cube root of the errors at the concluding places. Naturally, the question arises: why cube root? Since the components of the error vector $E_k$ are bounded in $[0, 1]$, the cube roots increase the error level for small signals maintaining their signs. The threshold adaptation process is continued so long as the error_sum does not converge within a permissible limit and the number of complete training cycles $r$ remains within a prescribed limit $r_{\text{max}}$.

The training algorithm with $n$ input–output patterns is presented next.

**Procedure Train_FPN** ([trainingpatterns]$[N_0, D, Th]$)

begin

\[ r := 0; \]

repeat

\[ S_1 := 0; S_2 := 0; \ldots; S_5 := 0; \text{Sum} := 0; \]

for $i := 1$ to $n$ do

begin

for $k := (r + 1)$ to $(r + \text{no-of-layers})$ do

begin

\[ [N_i(k)]_i := [N_0]_i \vee [P \circ \{\vee\{(Q \circ N_i)^c - Th\}\}] \]

\[ [E_k]_i := D_i - A \circ [N_i(k)]_i; \]

for $j := 1$ to $\text{number_of_components_of } [E_k]_i$ do

begin

\[ X_j := ([E_k(j)]_i)^{1/3}; \]

end

end

end

end

end

end.
\[
S_j := S_j + X_j; \quad e_j := [S_j]^3; \quad ||e_j|| = j\text{th component of composite error vector} \quad E'_k
\]
end for;
end for;

\[
Th := Th - W_{(k-r \times l)} \circ M^T \circ E'_k
\]
where \( M := P \circ (Q \circ P)^{(k-r \times l)} \)

\[E'_k \]
and \( W_{(k-r \times l)} = \begin{bmatrix} [\phi] \\ [1] \\ [\phi] \end{bmatrix}_{m \times s} \)

end for;
for \( j := 1 \) to \( \text{number\_of\_components\_of} \ E'_k \) do
begin
\[
\text{Sum} := \text{Sum} + S_j;
\]
\[
\text{PI}_r := \text{Sum}; \quad r := r + 1;
\]
end for;
until \( \text{PI}_{r-1} \leq \text{PI}_r \);
end.

For \( n \) sets of input–output patterns in procedure \text{Train\_FPN}, we determine \( [E_k]_i \) for \( i = 1 \) through \( n \) and consider the effects of the individual error vector \( E_k \) to construct a new error vector \( E'_k \). The \( j \)th component of \( E'_k \) is obtained as the cube of the sum of the cube-roots of the individual \( j \)th components of \( [E_k]_i \) for \( i = 1 \) through \( n \). The \( E'_k \) is then used to adapt the thresholds by the previous procedure. A performance index (PI) that attempts to minimize the sum of the components of \( E'_k \) is employed as the termination criterion of the procedure.

4. Analysis of convergence

The proposed algorithm converges to a stable point at the origin.

\textbf{Theorem 1.} \( \text{The error vector } E_k \text{ unconditionally converges to the origin in error-space.} \)

\textbf{Proof.} The error vector \( E_k \) is given by

\[
E_k = D - A \circ N_1(k) = D - A \circ N_0 \bigvee_{i=1}^{l} \left[ P \circ \left\{ \bigvee \{ (Q \circ N_i)^c - \text{Th}_k \} \right\} \right] \quad (10)
\]

where \( \text{Th}_k = \text{Th}_{k-1} - W_{(k-r \times l)} \circ M^T \circ E_k \) with

\[
M = P \circ (Q \circ P)^{(k-r \times l)} \quad \text{and} \quad W_{(k-r \times l)} = \begin{bmatrix} [\phi] \\ [1] \\ [\phi] \end{bmatrix}_{m \times s}
\]
and $\mathbf{Th}_k$ is the threshold vector $\mathbf{Th}$ obtained after updating thresholds at $k - 1$ number of layers (all of which need not be distinct).

We have

$$E_k = D - A \circ \left[ N_0 \lor \left[ P \circ \left\{ \bigvee_{i=1}^l \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right\} + W_{(k-r \times l)} \circ M^T \circ E_k \right] \right]$$

$$E_{k-1} = D - A \circ \left[ N_0 \lor \left[ P \circ \left\{ \bigvee_{i=1}^l \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right\} \right] \right]$$

Now,

$$\Delta E_{k-1} = E_k - E_{k-1} = A \circ \left[ N_0 \lor \left[ P \circ \left\{ \bigvee_{i=1}^l \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right\} \right] - A \circ \left[ N_0 \lor \left[ P \circ \left\{ \bigvee_{i=1}^l \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right\} \right] + W_{(k-r \times l)} \circ M^T \circ E_k \right] \right]$$

$$\Delta E_{k-1} = A \circ \left[ N_0 \lor [P \circ \left\{ (Q \circ N_j^c)^c - \mathbf{Th}_{k-1} \right\}] - A \circ \left[ N_0 \lor \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right] \right]$$

$$\Delta E_{k-1} = A \circ \left[ N_0 \lor [P \circ \left\{ (Q \circ N_j^c)^c - \mathbf{Th}_{k-1} \right\} - A \circ \left[ N_0 \lor \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right] + W_{(k-r \times l)} \circ M^T \circ E_k \right] \right]$$

$$\Delta E_{k-1} = A \circ \left[ N_0 \lor (P \circ y) - A \circ \left[ N_0 \lor \left\{ P \circ (y + W_{(k-r \times l)} \circ M^T \circ E_k) \right\} \right] \right]$$

where

$$y = (Q \circ N_j^c)^c - \mathbf{Th}_{k-1}$$

Substituting (12) in (11), we have

$$\Delta E_{k-1} = A \circ \left[ N_0 \lor \left\{ (Q \circ N_j^c)^c - \mathbf{Th}_{k-1} \right\} \right] - A \circ \left[ N_0 \lor \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right]$$

$$\Delta E_{k-1} = A \circ \left[ N_0 \lor \left\{ (Q \circ N_j^c)^c - \mathbf{Th}_{k-1} \right\} \right] - A \circ \left[ N_0 \lor \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right] + W_{(k-r \times l)} \circ M^T \circ E_k \right] \right]$$

$$\Delta E_{k-1} = A \circ \left[ N_0 \lor \left\{ (Q \circ N_j^c)^c - \mathbf{Th}_{k-1} \right\} \right] - A \circ \left[ N_0 \lor \left\{ (Q \circ N_i^c)^c - \mathbf{Th}_{k-1} \right\} \right]$$
Four possible cases are now considered.

Case I

\[ P \circ y \geq N_0 \quad \text{and} \quad (14) \]
\[ P \circ (y + W_{(k-rxl)} \circ M^T \circ E_k) \geq N_0 \quad (15) \]

We now present two extreme cases of analysis.

Case I(a)

Let the signs of all components of \( E_k \) be positive. Then

\[ \Delta E_{k-1} = A \circ P \circ y - A \circ P \circ (y + W_{(k-rxl)} \circ M^T \circ E_k) \]
\[ \geq - A \circ P \circ W_{(k-rxl)} \circ M^T \circ E_k \]
\[ \text{[since } X \circ Y + X \circ Z \geq X \circ (Y + Z)\]}
\[ = - [z_{ij}] \circ E_k, \text{say, where } [z_{ij}] = A \circ P \circ W_{(k-rxl)} \circ M^T \]

Since elements \( z_{ij} \) are bounded in the interval \( 0 \leq z_{ij} \leq 1 \), for any component of \([z_{ij}] \circ E_k \) close to the positive maximum (one), the corresponding component of \( \Delta E_{k-1} \) is \( \geq -1 \), which corresponds to convergence.

Case I(b)

Let the signs of all components of \( E_k \) be negative. Then

\[ \Delta E_{k-1} \leq - A \circ P \circ W_{(k-rxl)} \circ M^T \circ E_k \]
\[ \text{[since } X \circ Y + X \circ Z \leq X \circ (Y + Z)\]}

Now, when any component of \([z_{ij}] \circ E_k \) is the negative maximum \((-1 + \varepsilon)\), for a pre-assigned small positive number \( \varepsilon \), the corresponding component of \( \Delta E_{k-1} \) is \( \leq 1 - \varepsilon \), which too forces the system to converge.

When \( E_k \) contains both positive and negative components, the analysis could be carried out easily by point-wise formulation of the expressions for change of error at the output places in the network.

Case II.

When

\[ N_0 < P \circ y \quad \text{and} \quad (16) \]
\[ N_0 > P \circ (y + W_{(k-rxl)} \circ M^T \circ E_k) \quad (17) \]
\[ \Delta E_{k-1} = A \circ P \circ y - A \circ N_0 \geq 0 \text{ (the null vector).} \]

Since all components of \( E_k \) are negative [vide expressions (16) and (17)], \( \Delta E_{k-1} \geq 0 \) proves the convergence of \( E_k \).
Case III. When
\[ N_0 > P \circ y \quad \text{and} \]
\[ N_0 < P \circ (y + W_{(k-r \times l)} \circ M^T \circ E_k) \]
\[ \Delta E_{k-1} = A \circ N_0 - A \circ P \circ (y + W_{(k-r \times l)} \circ M^T \circ E_k) \leq 0 \] [vide (18) and (19)].

Since \( E_k > 0 \) and \( \Delta E_{k-1} < 0 \), \( E_k \) converges.

Case IV. When
\[ N_0 > P \circ y \quad \text{and} \]
\[ N_0 > P \circ (y + W_{(k-r \times l)} \circ M^T \circ E_k) \]
\[ \Delta E_{k-1} = 0, \] and thus \( E_k \) attains a constant value.

But in view of Cases I, II and III this constant value will be the null vector. Thus \( E_k \) unconditionally converges to the origin from any initial value in the error space. □

5. Application in fuzzy pattern recognition

This section illustrates an application of the proposed learning algorithm in fuzzy pattern recognition. As a case study we consider the problem of recognizing 2D objects such as circles, rings, ellipses, rectangles, and hexagons (Fig. 1) from their geometric features. The features considered here include area, perimeter, maximum length along the \( x \)-axis, maximum length along the \( y \)-axis, and inverse sphericity. The features have been selected from the point of view of their relative independence and totality, so that the feature-count is as low as possible but the features together serve as a sufficient descriptor of the objects. Since the measurements of these features may not be free from noise (measurement imprecision), we fuzzify the measurements and train a judiciously selected neural Petri net (Fig. 2) to recognize similar objects.

For fuzzification of features we could use typical fuzzy membership functions, such as triangular or Gaussian shaped functions. We, however, selected the following function, which is widely used in communication engineering for its inherent characteristics of mapping small variations in \( x \) to smaller variations in \( y \), and very large variations in \( x \) to relatively small variations in \( y \). Further, the function has a single control parameter that is used for scaling its \( x \)-range to make it suitable for specific applications:
\[ y = \log_{10}(1 + 5x/y) \]
\[ \frac{\log_{10}(1 + 5x)}{\log_{10}(1 + 5x)} \]

where \( x \) denotes the measured value of a feature, \( y \) represents the fuzzified membership value of \( x \), and \( \gamma \) denotes a normalizing factor, selected based on the range of \( x \).

The first four features (area of cross-section, perimeter, maximum \( x \)-length and maximum \( y \)-length) are fuzzified using the above membership function. Inverse sphericity is computed as the ratio of the perimeter of the object and the square root of its approximate cross-sectional area, and is fuzzified by subtracting a value of 3.8 from its estimated value. The subtraction is needed to keep the membership in \([0, 1]\).

The value of \( \gamma \) in expression (22) for the first four features has been determined experimentally, so as to keep the range \( y \) of the function in \([0, 1]\). Table 1 presents the selected values of \( \gamma \) for the respective features.

After the fuzzification of the five features of each object, we construct training instances for each object. Table 2 contains the training instances. It is prepared for training the FPN shown in Fig. 2. The columns under \( p_1, p_2, \ldots, p_5 \) in this table represent the fuzzified features: area, perimeter, maximum \( x \)-length, maximum \( y \)-length and inverse sphericity, respectively. The columns corresponding
Fig. 2. The proposed fuzzy Petri net (multiple copies of input places are shown for neatness).

### Table 1
Values of $\gamma$

<table>
<thead>
<tr>
<th>Index no.</th>
<th>Feature</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Area of cross-section</td>
<td>$10^4$</td>
</tr>
<tr>
<td>(2)</td>
<td>Perimeter</td>
<td>$6 \times 10^2$</td>
</tr>
<tr>
<td>(3)</td>
<td>Maximum $x$-direction length</td>
<td>135</td>
</tr>
<tr>
<td>(4)</td>
<td>Maximum $y$-direction length</td>
<td>135</td>
</tr>
</tbody>
</table>

### Table 2
Training instances

<table>
<thead>
<tr>
<th>Object</th>
<th>Places</th>
<th>Input vector components at</th>
<th>Target vector components at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$p_1$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>Circle</td>
<td>0.7728</td>
<td>0.6254</td>
<td>0.7407</td>
</tr>
<tr>
<td>Ring</td>
<td>0.5539</td>
<td>0.9237</td>
<td>0.8889</td>
</tr>
<tr>
<td>Ellipse</td>
<td>0.6503</td>
<td>0.5612</td>
<td>0.5333</td>
</tr>
<tr>
<td>Rectangle</td>
<td>0.6961</td>
<td>0.7251</td>
<td>0.8889</td>
</tr>
<tr>
<td>Hexagon</td>
<td>0.5363</td>
<td>0.5132</td>
<td>0.4444</td>
</tr>
</tbody>
</table>
to places $p_{12}, p_{13}, \ldots, p_{16}$ denote the membership of the recognized objects: circle, ring, ellipse, rectangle and hexagon, respectively. The minimum acceptable membership for an object to be recognized correctly is (arbitrarily) set at 0.41. After the network of Fig. 2 is trained with the instances prescribed in Table 2, we can use the network for recognizing an unknown object.

The FPN in Fig. 2 contains three layers: (a) one input layer consisting of places $p_1$ through $p_5$, (b) one hidden layer comprising transitions $t_{r1}, t_{r2}, \ldots, t_{r8}$ followed by places $p_6, p_7, \ldots, p_{11}$ and (c) one output layer comprising transitions $t_{r9}, t_{r10}, \ldots, t_{r13}$ followed by concluding places $p_{12}, p_{13}, \ldots, p_{16}$. The five fuzzified features are mapped at the input places $p_1, p_2, \ldots, p_5$. The places $p_{12}, p_{13}, \ldots, p_{16}$ in the output layer correspond, respectively, to the five objects—circle, ring, ellipse, rectangle and hexagon.

The most important aspect in the construction of the FPN of Fig. 2 is the selection of transitions and places in the hidden layer. The inputs of the transitions $t_{r1}, t_{r2}, \ldots, t_{r3}$ in the hidden layer are selected by taking into account the logical joint occurrence of the features. For example, large area and perimeter should co-exist in circular patterns. So, both $p_1$ and $p_2$ are considered as input places of transition $t_{r2}$. Further, to emphasize the importance of area (see Table 2, column under $p_1$), we use transition $t_{r1}$. To recognize circles, $p_6$ is considered as the output place of both $t_{r1}$ and $t_{r2}$. Recognition of circles can thus be accomplished by considering $p_6, p_3$ and $p_4$ as the input places of $t_{r9}$, and $p_{12}$ as the output place of $t_{r9}$. Since inverse sphericity is small for circles (see Table 2), we do not use $p_5$ for recognition of circles.

In principle, both transitions and places in the hidden layers are selected by analyzing the influence of co-existence and independence of features. For each set of joint features, we need one transition. The places in the hidden layer are selected to emphasize one independent feature (or a set of joint features) over one or several other features. The semantic interpretation of the places in the hidden layer is explained below:

- $p_6$: objects having higher priority on large area than the joint occurrence of large area and large perimeter.
- $p_7$: objects having higher priority on large perimeter than the joint occurrence of large area and large perimeter.
- $p_8$: objects having higher priority on maximum $y$-length than the joint occurrence of large area, large perimeter and inverse sphericity.
- $p_9$: objects having higher priority on inverse sphericity or maximum $x$-length than the joint occurrence of maximum $x$-length and maximum $y$-length.
- $p_{10}$: objects having priority on maximum $x$-length and maximum $y$-length.
- $p_{11}$: objects having priority on maximum $y$-length and inverse sphericity.

The threshold values generated by the training algorithm are presented in Table 3.
<table>
<thead>
<tr>
<th>Transition no.</th>
<th>tr₁</th>
<th>tr₂</th>
<th>tr₃</th>
<th>tr₄</th>
<th>tr₅</th>
<th>tr₆</th>
<th>tr₇</th>
<th>tr₈</th>
<th>tr₉</th>
<th>tr₁₀</th>
<th>tr₁₁</th>
<th>tr₁₂</th>
<th>tr₁₃</th>
<th>tr₁₄</th>
<th>tr₁₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold value</td>
<td>0.0613</td>
<td>0.2104</td>
<td>0.0950</td>
<td>0.0574</td>
<td>0.0000</td>
<td>0.3353</td>
<td>0.0217</td>
<td>0.0000</td>
<td>0.3229</td>
<td>0.4185</td>
<td>0.0527</td>
<td>0.2860</td>
<td>0.2782</td>
<td>0.8716</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
The training cycle requires 157 iterations with an estimated error margin of $10^{-4}$.

A plot of the performance index $PI_r$ for the given three-layered network (Fig. 2) for 78 complete training cycles is presented in Fig. 3.

After the training of the network is over, we can use the network for application in object recognition. We experimented with various unknown objects, and obtained interesting results. For example, when the fuzzified feature vector for a regular pentagon was supplied at the input of the pre-trained neural FPN, the network classified it to its nearest class (hexagon). When the fuzzified feature vector of a $90^\circ$ rotated ellipse (see the unknown sample in Fig. 1) was supplied as the input of the pre-trained network, it correctly classified the object as an ellipse (see Table 4). It is interesting to note that the “unknown” ellipse pattern is definitely different from the training instance ellipse pattern as the two patterns differ in their maximum $x$-length and maximum $y$-length features.

6. Conclusions

This paper presented a new learning model of neural net capable of representing the semantics of high-level knowledge. The unconditional convergence of the error states to the origin in error space has been proved and is an added advantage of the proposed learning model. The model has successfully been applied to a practical problem in fuzzy pattern recognition. The generic scheme of the model should find applications in many-to-many fuzzy semantic function realization as well as in recognition of objects from their fuzzy feature spaces.
Table 4
Input/output vector components for an unknown object

<table>
<thead>
<tr>
<th>Object</th>
<th>Places</th>
<th>Input vector components at</th>
<th>Output vector components at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Area</td>
<td>Perimeter</td>
</tr>
<tr>
<td>Unknown object</td>
<td></td>
<td>0.6046</td>
<td>0.5474</td>
</tr>
</tbody>
</table>
Acknowledgement

The authors gratefully acknowledge the helpful comments and criticisms of four anonymous referees.

References